WO02087235

Publication Title:

SYSTEM FOR APPLYING METRIC TO MULTIMEDIA FILES OVER NETWORK

Abstract:

Systems and methods for streaming of multimedia files over a network are described. A streaming delivery accelerator (SDA) (28, Fig. 2) receives content from a content provider (16, Fig. 2) and streams the received content to a user (12, Fig. 2). Content is streamed when a quality metric exceeds a predetermined value. A similar quality metric applies to caching and storing the received content in the SDA' cache. Methods for effective cache management are also described. The SDA can separate (Shred) the content into contiguous cache fields suitable for streaming. Also, checksums can migrate from the content file to the shredded cache fields and between different network protocols without the need for recomputing the checksums. The caching process can be iterative, with only content not previously retained in the SDA's cache being requested from the retransmitted by the content provider. The SDA can be disconnected from the content provider when sufficient content for a predetermined time of play has been received. A method for cache eviction of content no longer of interest to users is also described.

Data supplied from the esp@cenet database - http://ep.espacenet.com

This Patent PDF Generated by Patent Fetcher(TM), a service of Patent Logistics, LLC

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
31 October 2002 (31.10.2002)

PCT

(10) International Publication Number
WO 02/087235 A1

(51) International Patent Classification[7]: H04N 7/10, G06F 12/00

(21) International Application Number: PCT/US02/12430

(22) International Filing Date: 19 April 2002 (19.04.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/284,973 19 April 2001 (19.04.2001) US
60/289,409 8 May 2001 (08.05.2001) US

(71) Applicant: VIVIDON, INC. [US/US]; 142 North Road, Sudbury, MA 01776 (US).

(72) Inventors: CATES, Joshua, Ian; 115 Harvard Street #4, Cambridge, MA 02139 (US). HUNT, Russell; 7100 Westworth Drive, Willow Springs, NC 27592 (US). KESSLER, Garry, Kenneth; 20 Ruggles Street, Westboro, MA 01581 (US). PICKNEY, Thomas, III; 102 Hancock Street #2, Cambridge, MA 02139 (US). PROVENZANO, Christopher, Angelo; 95 Liberty Avenue, Somerville, MA 02144 (US). SCHLOWSKY-FISCHER, Mark, Harold; 25 Curtis Road, Apt. 1, Somerville, MA 20144 (US). THOMAS, Benjamin, Joseph, III; 2 Appletree Lane, Bedford, MA 01730 (US). WYATT, Douglas, Karl; 22 Francesca #2, Someville, MA 02144 (US).

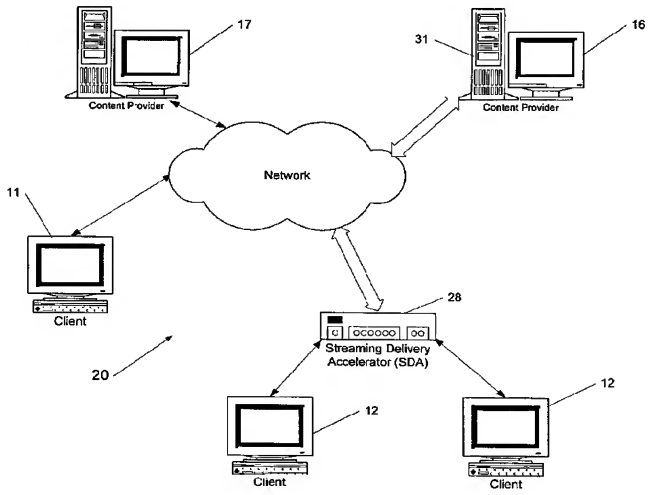
(74) Agents: KELLY, Edward, J. et al.; Ropes & Gray, Patent Group, One International Place, Boston, MA 02110 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: SYSTEM FOR APPLYING METRIC TO MULTIMEDIA FILES OVER NETWORK



(57) Abstract: Systems and methods for streaming of multimedia files over a network are described. A streaming delivery accelerator (SDA) (28, Fig. 2) receives content from a content provider (16, Fig. 2) and streams the received content to a user (12, Fig. 2). Content is streamed when a quality metric exceeds a predetermined value. A similar quality metric applies to caching and storing the received content in the SDA' cache. Methods for effective cache management are also described. The SDA can separate (Shred) the content into contiguous cache fields suitable for streaming. Also, checksums can migrate from the content file to the shredded cache fields and between different network protocols without the need for recomputing the checksums. The caching process can be iterative, with only content not previously retained in the SDA's cache being requested from the retransmitted by the content provider. The SDA can be disconnected from the content provider when sufficient content for a predetermined time of play has been received. A method for cache eviction of content no longer of interest to users is also described.

WO 02/087235 A1

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM FOR APPLYING METRIC TO MULTIMEDIA FILES OVER NETWORK

Field of the Invention

The invention is directed to real-time streaming of multimedia files over a network. More particularly, the invention is directed to a method for caching files with selected streaming characteristics from a content provider employing a quality metric for the cached content and to a multimedia streaming delivery accelerator (SDA) that caches the content and produces cache files meeting the quality metric for playback to an end-user. The invention is also directed to a method for efficiently allocating and managing the allocation of memory for file streaming and to an SDA with persistent and buffer memory adapted to efficiently store and stream the cache files.

Background of the Invention

The Internet has witnessed a rapid growth in the deployment of Web-based streaming applications during recent years. In these applications, congestion control and quality adaptation is paramount so as to match the stream quality delivered to an end-user to the average available bandwidth. In other words, the delivered quality is limited by the bottleneck bandwidth on the path to the end-user. Moreover, there is a need for scalability as the number of people accessing multimedia services over the Internet grows, which is further exacerbated by the rapidly increasing demand for bandwidth-intensive video and audio streaming services. Adding more bandwidth and Quality-of-Service (QoS) support to the Internet is one potential remedy for performance problems, but large scale deployment is costly and may take a long time.

More recently, content providers have began to offer solutions encompassing technologies such as caching, enhanced routing and load balancing. These solutions do not require any specific support from the network, but provide the end-user with an improved experience to due the enhanced network efficiency.

However, there is still a need for improvement of delivery of streaming multimedia files over a network, in particular over the Internet, and more particularly for a system and method that can efficiently deliver both scheduled and on-demand streamed content to an end-user over variable bandwidth connections.

Summary of the Invention

The invention is directed to caching content in a network appliance, such as a Streaming Delivery Accelerator (SDA), and streaming the cached content to a user requesting the content. The SDA applies a quality metric to the content received from a content provider and streams the cached content to a user and/or caches and stores the content if the quality metric exceeds a quality threshold value. The quality values for caching/storing and streaming need not be the same.

According to one aspect of the invention, a method of producing a cache file includes receiving a content file from a content provider; defining a quality threshold value, wherein the quality threshold value represents an number of packets representative of the content file; defining a quality metric of the received content file, wherein the quality metric represents a number of received packets out of the number of packets representative of the content file; and testing the quality metric of the received content against a first quality threshold value to determine if the quality metric is equal to the a first quality threshold value. If the quality metric is equal to the first quality threshold value, then the received content is cached from a content provider and a cache file is produced. Conversely, if the quality metric is smaller than the first quality threshold value, then the cached content is discarded.

According to another aspect of the invention, a method of producing a cache file includes receiving a content file from a content provider; defining a quality threshold value of the content file, the threshold value representing a number of packets representative of the content file; defining a quality metric of the received content, the quality metric representing a number of the received packets out of the number of packets representative of the content file; and caching content from the content provider if the quality metric of the cached content equal

to the quality threshold value. The method further includes streaming the cached content to a user; and if the user interrupts streaming, continuing caching a remaining portion of the content if the cached content represents at least a predetermined fraction of the content file; and deleting the cached content if the cached content represents less than the predetermined fraction of the content file.

Another aspect of the invention relates to a streaming delivery accelerator (SDA) for producing a cache file. The SDA includes at least one input channel that receives content from a content provider; a cache memory that stores a the received content; at least one output channel with a predetermined bandwidth for streaming a cache file generated from the cached content to a user. The received content has a quality metric, which represents a number of the received packets out of a number of packets representative of the content file. The received content is stored in cache memory if the quality metric of the received content is at least equal to a quality threshold value.

Additional embodiments of the invention may include one or more of the following features. The cache file may be produced by assembling the packets of the cached content as a contiguous cache file having a predetermined order of packets, and storing the contiguous cache file. When testing the quality metric, the received content may first be associated with a second quality metric with a second quality threshold value, wherein although the received content may have a lesser quality than desired, the received content is transmitted to the client if the second quality metric is greater than second quality threshold value, and the SDA continues caching content from the content provider until the first quality metric is greater than first quality threshold value. In this way, a first user can receive the content, albeit imperfect, with subsequent users enjoying an improved quality. The first and second quality threshold values can be identical or different. Caching can be iterative, wherein packets that are missing from the content can be requested from the content provider, and the stored content is updated with the received packets until the first quality metric of the updated cache file is equal to the first quality threshold value.

The quality threshold value can also refer to a required transmission order of the cached packets, so that the quality metric can be tested by determining an actual order of the received packets, for example, a time-sequential order.

The cache file can also be produced by "shredding" the source content file into at least one shredded cache file having a streaming characteristic, for example, a bit rate, and storing the at least one shredded cache file. Multiple files can be shredded from a single source content file. Checksums for packets of the received content can either be sent by the content provider or computed by the SDA from the received content. The checksums can be stored in the SDA's memory and associated with the packets in the cache file to be streamed to a user. The cache file with the checksums can be multicasted to a plurality of clients. The network protocol information can be removed from the cache file and the cache file can be stored as a protocol-independent canonical file.

The invention is also directed to an SDA that shreds the content file into contiguous files suitable for streaming to an end-user. The contiguous files can have different content and/or transmission rates. Such preassembled files greatly enhance the overall efficiency of the SDA.

According to one aspect of the invention, a method for shredding a content file to create a contiguous cache file for rapid streaming delivery includes receiving a content file from a content provider, identifying content file packets to be associated with the cache file, and assembling the identified content file packets into the contiguous cache file having a predetermined streaming characteristic. The contiguous cache file can be stored on a persistent storage medium.

According to another aspect of the invention, a streaming delivery accelerator (SDA) is disclosed that shreds a content file to create a contiguous cache file for rapid streaming delivery. The SDA includes at least one input channel that receives a content file from a content provider, at least one output channel with a predetermined bandwidth for streaming the contiguous cache file to a user, and a shredder that identifies content file packets to be associated with the contiguous cache file and assembles the identified packets into the contiguous cache file having a predetermined streaming characteristic. The SDA further

includes a persistent storage device that stores the contiguous cache file.

Embodiments of the invention can include one or more of the following features.

The shredder may provide a pointer to an identified content packet and assembles the contiguous cache file by fetching from the received content file the content packets identified by the pointers. In this way, duplicate packets need not be cached and stored multiple times. The identified packets in the contiguous cache file may be stored as an ordered sequence of the packets, which can correspond to a time-sequential order or a transmission order of the packets to the user. Advantageously, a plurality of contiguous files with different streaming characteristics, such as different content or transmission rates, may be produced and stored. The contiguous file may be generated by the shredder concurrent with a streaming request received from a user and/or by the same process that produces the file being streamed to the user.

The content file can be received from the content provider via a first network protocol, and the streaming characteristic can include transmission of the contiguous file to a user via a second network protocol that is either different from or the same as the first network protocol. This is advantageously done by removing information about the first network protocol, such as header information, and storing the contiguous file in protocol-independent canonical form. Likewise, protocol information can be added when the cache file is streamed to the user. Exemplary network protocols can be any of MMST (Microsoft Media Server (TCP)), MMSU (Microsoft Media Server (UDP)), MMSH (Microsoft Media Server (HTTP)), SSH/SCP (secure shell, secure copy), HTTP (hypertext transport protocol), FTP (file transfer protocol), RTSP/RTP/RDT (Real Time Streaming Protocol/ Real-time Transport Protocol/ Real Data Transport) and PNA (Progressive Networks Audio), PNM (Progressive Networks Media) or a combination thereof.

Checksums can be either received from the content provider with the packets or computed from the received packets. These checksums are referenced to the identified packets and stored, obviating the need to recalculate checksums,

except for simple concatenation, when the cache file is streamed to the user.

The invention is further includes directed to an SDA with a persistent memory device, such as a disk, and at least two volatile buffer memory devices which cooperate to provide a contiguous and uninterrupted data stream of the content to the user. The first buffer memory caches content from disk for streaming. A second buffer memory starts caching content from the disk memory before the first buffer memory finishes streaming content to the user. Thereafter, the roles of the first and second buffer are reversed. The SDA can thereby continuously stream the content by fetching content from the second buffer when the first buffer is empty and vice versa. This can prevent an interruption of the bit stream to the user due to disk seeks.

According to one aspect of the invention, a method of allocating data blocks for streaming a file includes determining a data transfer characteristic for streaming the file, allocating data blocks having a block size on a persistent storage medium, wherein the block size is related to the data transfer characteristic, and storing the file in form of the data blocks on the persistent storage medium. The method further includes transferring the data blocks from the persistent storage medium into a first buffer memory, wherein the size of the data blocks in the first buffer memory is identical to the block size, and based on an actual bit rate determined for the stream to a user, allocating a second buffer memory to receive from the persistent storage medium additional data blocks based on the actual bit rate to a user, wherein the size of the data blocks in the second buffer memory is also identical to the block size. The additional data blocks are then transmitted from the persistent storage medium to the second buffer memory when an estimated transmit time for streaming a portion of the data blocks remaining in the first buffer to the user is approximately equal to a time required to transfer the additional data blocks from the persistent storage medium into the second buffer.

According to another aspect of the invention, a device for allocating data blocks for streaming a file includes a persistent storage medium storing the data blocks of the file, wherein the data blocks have a block size allocated related to a data transfer characteristic. The device further includes a first buffer memory that

receives the data blocks from the persistent storage medium, wherein the size of the data blocks in the first buffer memory is identical to the block size, and a second buffer memory that receives additional data blocks from the persistent storage medium, wherein the size of the data blocks in the second buffer memory is also identical to the block size. The additional data are received in the second buffer memory when an estimated transmit time for streaming a portion of the data blocks remaining in the first buffer to the user is approximately equal to a time required to transfer the additional data blocks from the persistent storage medium into the second buffer.

According to yet another aspect of the invention, a streaming delivery accelerator (SDA) for efficiently streaming a file, for example over a network, includes at least one input channel that receives a content file from a content provider, at least one output channel with a predetermined streaming characteristic for streaming a cache file to a user and a persistent storage medium that caches data blocks of the content file and stores the data blocks as the cache file, said data blocks having a block size allocated related to a data transfer characteristic. The SDA further includes a first buffer memory that receives the data blocks from the persistent storage medium, said data blocks in the first buffer memory having the same block size as in the persistent storage medium, and a second buffer memory that receives additional data blocks from the persistent storage medium, said data blocks in the second buffer memory also having the same block size as the block size as in the persistent storage medium. The additional data are received in the second buffer memory when an estimated transmit time for streaming a portion of the data blocks remaining in the first buffer to the user is approximately equal to a time required to transfer the additional data blocks from the persistent storage medium into the second buffer.

Embodiments of the invention can include one or more of the following features. The SDA can store files of different streaming bit rates, so that the transfer characteristic can be a maximum bit rate for the file. The persistent storage medium can be a disk storage medium, such as a magnetic, optic or magneto-optic disk medium, whereas the first buffer memory and the second buffer memory are volatile memory devices, such as random access memory (RAM).

The blocks can have different block sizes, which can be determined by a maximum streaming rate of the file, which can be greater than the streaming (bit) rate to the user, and/or a maximum seek time for the data blocks on the disk storage medium. The blocks and/or block sizes can be organized in an inode structure, wherein the blocks have at least two block types. A first block type is associated with a direct block which includes the data blocks and a second block type with an indirect block that has a pointer to a other direct blocks or of additional indirect blocks. The block size of the additional direct blocks can be greater than the block size of the indirect blocks and the blocks of the first block type. The persistent storage medium stores at least the direct blocks as a contiguous sequence of blocks.

The at least one input channel can operate using a first network protocol, and the at least one output channel operates using a second network protocol that can be identical to or different from the first network protocol. The cache file can be stored in the persistent medium device as a protocol-independent canonical file. The data blocks of the stored cache file can include checksum representative of payload data, which obviates the need to recomputed the checksums each time the file is streamed. In most cases, checksums of packets of the streamed file may be computed by simply adding the stored checksums or portions thereof.

The invention is also directed to efficiently caching content in an SDA and to evicting content that is no longer of interest to users from the SDA's cache. According to one aspect of the invention, a method for allocating cache memory for streaming includes receiving a request from a user for streaming content from a content file, wherein the user indicates a play position; checking if the content is in cache memory, and if the content is not in memory, caching the content at least at the play position from a content provider. The method further includes determining a cache fill initiate horizon (CFIH), which defines a cache memory content sufficient to begin streaming to the user, and caching content from the content provider that is missing between the play position and the CFIH. Also defined is a cache fill terminate horizon (CFTH), which defines additional cache memory content sufficient for maintaining a content stream of predetermined duration after streaming to the user has begun. The CFIH and CFTH are synchronously advanced

during play; and content from the content provider that is missing between the CFIH and the CFTH is cached.

Embodiments of the invention may include one or more of the following features. The settings of the CFIH and the CFTH can depend on a transmission rate of the streamed content to the user and/or a header duration information of the content file, such as the nominal bit rate of the content file. The connection between the SDA and the content provider can be severed when the CFTH reaches an end of file (EOF) position and/or when the content between the play position and the CFIH is in the cache memory. The setting of the CFIH and the CFTH can also depend on a time for establishing a connection to the content provider.

If an interruption in streaming the content to the user is detected, it is determined if the cache memory contains more than a predetermined fraction of the content file. If the cache memory contains more than the predetermined fraction of the content file, the remaining fraction of the content file is still cached from the content provider. Conversely, if the cache memory contains less than the predetermined fraction of the content file, the cached content is purged from cache memory.

According to another aspect of the invention, a method of managing cache memory for streaming includes receiving from a content provider content of a content file; caching the content and storing at least a portion of said content as a cache file in a cache memory for streaming; collecting a request history for streaming of the cache file; determining from the request history the portion of the cache file that meets predetermined cache eviction criterion; and evicting from the cache memory the determined portion of the cache file that meets the cache eviction criterion.

Embodiments of the invention may include one or more of the following features. The cache eviction criterion may include a fixed time period, an elapsed time since a previous request, a frequency of previous requests, and a quality metric; the cache eviction criterion may also apply to a location of the cached content within a cache file, so that, for example, segments of the cache file located in the middle or at the end of the cache file can be preferably evicted. The content

can be stored in the cache file as payload data and as system data, such as metadata, wherein the payload data of the determined portion of the cache file are evicted from the cache memory before the system data of the determined portions are evicted.

Further features and advantages of the present invention will be apparent from the following description of preferred embodiments and from the claims.

Brief Description of the Drawings

The following figures depict certain illustrative embodiments of the invention in which like reference numerals refer to like elements. These depicted embodiments are to be understood as illustrative of the invention and not as limiting in any way.

Fig. 1 depicts a prior art system for streaming content over a network;

Fig. 2 depicts a system for streaming content over a network with a streaming delivery accelerator (SDA) and content and network management;

Fig. 3 is a schematic block diagram of an SDA cache architecture;

Fig. 4 is a flow diagram depicting a process for caching using a quality metric;

Fig. 5 schematically depicts a cache fill process;

Fig. 6 is a flow diagram depicting an initial cache fill process;

Fig. 7 is a flow diagram depicting a process for caching additional content ;

Fig. 8 shows schematically a cache filling and eviction process;

Fig. 9 shows schematically "shredding" of a source content file;

Fig. 10 is a schematic diagram of files subject to different cache eviction policies; and

Fig. 11 depicts disk storage with variable size file allocation units.

Best Mode(s) for Carrying Out the Invention

Detailed Description of Certain Illustrated Embodiments

The invention is directed to caching a content file from a content provider and streaming cached content/ and or a cached file to an end-user over a network. The end-user can be an individual subscriber and/or an enterprise, where several clients are connected, for example, via an Intranet, LAN or WAN.

The organization of the SDA and in particular the SDA's memory organization is described below with reference to Fig. 3. Fig. 4 is a schematic flow diagram of the quality caching process, with cache filling described in more detail with reference to Figs. 5-7. Fig. 8 and Fig. 10 depict aspects of a cache eviction process. Fig. 9 illustrates schematically the organization of the shredded files. Each of the shredded files can be manipulated independently. For example, one or more of the shredded files can be deleted from memory, while others are retained for future use. Fig. 11 illustrates schematically the organization of the persistent memory, for example the disk cache depicted in Fig. 3, which is organized in form of an inode structure known, for example, from the UNIX operating system. The size of the data blocks transferred from the disk to the volatile buffer memory is preferably identical to the size of the file allocation units on the disk.

Before describing the invention in detail, reference is made first to Fig. 1 to provide some background information. A conventional system 10 includes content provider servers 16, 17 providing content to end-users or client system 11, 12, with the servers 16, 17 being connected to the client system 12 through a network 14, such as the Internet or a LAN, which can support different connections, such as a telephone modem, IDSN, ATM, LAN, WAN, Ethernet, T1/T3, Frame Relay, Sonet, etc. To obtain content from a content provider 16, a client 12 will typically open a browser window and establish a connection to the content provider 16, for example, by clicking in the window on a link or http address. The content provider 16 can then transmit the content directly to the client 12.

In the depicted system 10, the client system 11, 12 can be any suitable computer system such as a PC workstation, a handheld computing device, a

wireless communication device, or any other such device, equipped with a network client capable of accessing a network server and interacting with server 16, 17 to exchange content with the servers 16, 17. The network client may be a web client, such as a web browser that can include the Netscape web browser, the Microsoft Internet explorer web browser, the Lynx web browser, or a proprietary web browser, or web client that allows the user to request and receive streaming content from the network server. The communication path between the clients 11, 12 and the servers 16, 17 can be an unsecured communication path, such as the Internet 14, or a secure communication path, for example, the Netscape secured socket layer (SSL) security mechanism that provides to a remote user a trusted path between a conventional web browser program and a web server.

This approach has several drawbacks. For example, separate communication channels need to be opened to connect several clients 12 to the content provider 16, even if the clients 12 request the same content. The content provider 16 may be at a distant location from the clients 12, so that the replication of connections would require excessive bandwidth, which can introduce latency and network congestion. Accordingly, these bottlenecks should be "smoothed" out, which is one of the tasks performed by the SDA described in detail below.

In an improved multimedia streaming solution 20 depicted in Fig. 2, the problems associated with the different transmission characteristics and pathway bandwidths are alleviated by placing a Streaming Delivery Accelerator (SDA) 28 intermediate between the content provider 16 and one or more clients 12. Although network 24 is shown as a single network, such as the Internet, network 24 can also be a local area network (LAN), an intranet or a combination thereof. Moreover, the connection between the SDA 28 and the clients 12 can also be a network connection, such as a LAN or an intranet, or even the Internet. In the streaming solution 20, the clients 12 no longer communicate directly with the content provider 16 when requesting content. Instead, client requests for streaming files from the content provider 16 are routed through and monitored by a Streaming Delivery Accelerator (SDA) 28. A service manager (not shown) interacting with software that can be connected to or embedded in the SDA system 28 can provide aggregate performance monitoring and alarm management on a network-wide

basis, as well as management/ configuration of system resources and protocols. Management operations can also be performed from a client 12 linked to the network 24 and running suitable software, for example, in conjunction with a Web browser.

In the exemplary streaming solution 20 depicted in Fig. 2, a client 12 requests content 31, for example a movie trailer, from a content provider 16. The client 12 has a network connection with a certain bandwidth, for example, via a modem or a T1/T3 connection. The user request is transmitted to the Streaming Delivery Accelerator (SDA) 28. The SDA 28 transmits the request for the specified file to the content provider 16. In many cases, the content 31 stored at the content provider 16, which can include video/audio files, html files, text files and the like, may not be in a format suitable for streaming directly to the client 12. For example, a file may be transmitted from the content provider 16 to the SDA 28 with a network protocol that is incompatible with the network protocol used by the client 12. Moreover, the application software at the client 12 may require interleaved video and audio, whereas the contents 31 stores video and audio as separate files. The SDA 28 should therefore be able to perform a protocol translation and/or cache and store files representing the content requested by the client in a protocol-independent (canonical) form. The term "network protocol" used in the following description is to be understood to include also "application protocols", which represent the set of protocols corresponding to protocol layers 4 through 7 inclusive in the ISO/OSI protocol model, which is incorporated herein by reference. Accordingly, these terms can be used interchangeably. Caching is defined as storing a copy of a stream-set for later playback. Protocol-independent caching will be described below.

Fig. 3 shows in form of a schematic block diagram the architecture of an exemplary SDA 28. The SDA 28 includes a protocol translator 36 which strip the network protocol headers from the received packets and generates protocol-independent canonical payload data packets. Also incorporated is a shredder 35 that is capable of selecting from received content those packets perceived as being of use for streaming to end users. These canonical packets are then written into the SDA's cache memory 32 which can include a disk cache 31 and one or more

buffer caches 33, 34. When files are streamed to clients 12, the canonical packets are retrieved from disk cache 31 and written to buffer 33 as a contiguous stream adapted to the streaming rate to the client 12. The SDA 28 includes another protocol translator 38 at the output, which appends the canonical packets with suitable network protocol headers for streaming to the client 12. The functionality of the optional second buffer cache 34 and its cooperation with the first buffer cache 33 and the disk cache 31 will be described later.

The data transmitted by the content provider 16 may be a superset of the data to be streamed to the client 12. The SDA 28 can then select from the data received from the content provider 16 those data, typically in the form of data packets, that correspond to the specific file requested by the client 12, and assemble these data into a contiguous file for streaming to the client 12 via network 24.

As indicated in Fig. 2, several clients 12 can be connected to the SDA 28 and may request the same content either simultaneously or at different times. Since SDA's 28 can be provided at various sites in a network, network traffic can be reduced substantially, if a client 12 can receive the requested file from an SDA 28 located in the vicinity of the client 12, for example, an SDA 28 located on the same intranet as the client 12, or from an SDA 28 that has little latency. A subsequent client request for the same file can then be satisfied by the SDA 28 without involvement of the content provider 16. An SDA 28 can also meet requests for multicasting, so that even if the content file was not previously cached by the SDA 28, only a single connection would be required between the SDA 28 and the content provider 16, with packet replication performed by the SDA 28.

There are at least three types of media files in use today within the streaming media server space that are used to support client requests: (1) a single rate, multi-stream file that can be composed of a video stream at a single bit-rate, an audio stream at a single bit-rate and optionally other stream types such as text, html or scripting; (2) a multi-bit-rate file that can include several video streams of differing bit-rates, audio and other stream types and can therefore service from the data stored within the file many different client requests with different transmission

rates; and (3) a direct stream capture which captures only the necessary data bits to support the requested stream. The SDA is designed to handle those different streams.

The terms "Quality Caching" and the metric associated with Quality Caching ("Quality Metric") are useful concepts for caching content. One measurement of the quality of a stream is the ratio of received packets at the SDA to the total number of packets representative of the file or the current segment thereof. Reasons for not receiving all packets, i.e., for a low quality metric, include but are not limited to: network congestion; the use of network transport that does not guarantee delivery of all packets; an end-user / client device signaling a content provider to reduce information being transferred due to the client's inability to process the received information; and/or an end-user signaling a content provider to pause, stop, rewind or fast forward the stream. The SDA can advantageously apply the quality metric of (received packets/total packets) in cases where the SDA knows up-front the total expected number of packets or if the SDA is able to detect that not all packets have been received. For example, the HTTP and MMS protocols indicate the number of expected bytes or packets that should be received. When bandwidth adaptation techniques are enabled (for example, in the MMS protocol), the SDA can also infer from missing delta-frames in a sequence of key frames of video that some packets are missing. The SDA software of the invention can hence determine whether it has received a sufficient percentage of packets to successfully serve the stream out of the cache.

An exemplary process 40 for caching content to serve the content to a client and for retaining useful content for subsequent client streaming requests is depicted in the schematic flow diagram of Fig. 4. When a client requests content, step 41, the process 40 checks, step 42, if the content or at least part of the content, is already in the cache. If any content is detected in the cache, it is checked in step 43 if there sufficient content for streaming to the client has been cached, for example, based on the aforedescribed quality metric and defined by a quality threshold value. In other words, step 43 checks if the quality metric in the cache is greater than a preset threshold value. If enough content is present, the cached

content is served to the client, step 44. Otherwise, the SDA requests and retrieves additional content, preferably the entire missing content, from the content provider to serve to the client, step 45. The actual quantity of the content to be cached in step 45 depends on the settings of the Cache fill initiate horizon (CFIH) and Cache fill terminate horizon (CFTH), which will be discussed below with reference to Figs. 5-7.

The process 40 will then check in step 46 if a sufficient percentage of the file has been cached to make it worthwhile to keep the cached file in the cache to satisfy future streaming requests from clients. If less than a preset percentage of the file (file threshold value) was cached, the cached content is discarded, step 48. Otherwise, the cached content is kept in the cache and a cache file is produced and stored in memory. The file threshold value should ideally be 100%, in which case the stream set is not cached if: 1) any packets are missing from the stream; and/or 2) a pre-fix of the packets from the stream is not received, which would make it impossible to determine its proper position in the stream. However, a lower file threshold value may be tolerated, depending for example on the network protocol and image compression used, if dropped or frozen sections of images are acceptable. A corresponding range of values applied to the quality threshold value. However, the file threshold value and the quality threshold value need not be identical.

An efficient way to obtain the remaining content is to incrementally fill in the incomplete cache stream sets from additional data received from the content provider, but cache only the packets that were missing from the cached stream set. These packets can be requested by the SDA and extracted from the source content file at the content provider. The latter approach is referred to as iterative caching and will now be discussed.

Iterative caching is the ability to incrementally improve the quality of a cached stream. The exemplary SDA 28 may have previously cached some but not all of a streamed file during a prior request. Subsequently, another request for the same streamed file is received by the SDA. The SDA then proceeds to fetch from

the content provider additional packets of the content to incrementally build up a complete set of data. It can be seen that successive requests will not degrade the quality of the subsets already residing in the cache.

Iterative caching is useful in situations where, for example: 1) there is intermittent connectivity between locations the SDA and a content provider or other content server; 2) there is low bandwidth connection between the SDA and the content provider or other content server; and 3) there is a large number of possible subsets of the content, only a few of which are useful at a particular client location. Iterative caching can be used in both streaming media caching and in network attached storage caching. Iterative caching becomes increasingly important as the space taken up by the data becomes very large.

Figs. 5 - 7 illustrate an exemplary iterative cache fill process at the SDA wherein an exemplary contiguous interleaved (video/audio) stream set 50 includes packets 0, 1, 2, .. , N, ... that are to be streamed to a user. As seen in Fig. 5, it will be assumed that packets 0, 1, 2, j, and k already reside in the SDA's memory, and that several packets 3, .., k, ..., N are missing. The packets can be indexed by packet index 52. When an end-user requests a streamed file starting at a play position P, a buffer memory is filled up to a cache fill initiate horizon (CFIH) which represents the minimum number of packets that should be present in order to obtain an initial play length of the stream with the desired quality metric. For example, all packets between packet 2 and j should be present before streaming to the user begins. With iterative caching, the cache is incrementally filled up to the cache fill initiate horizon (CFIH) by fetching from the content provider the missing packets 3, ..., j-1. The same iterative caching process applies to filling the cache up to cache fill terminate horizon (CFTH), wherein the number of packets between the CFIH and the CFTH correspond to an assumed viewing time for a user, which can be experience-based and may hence depend on the particular viewed content. It will be understood that the packet k can represent more than a single packet, i.e., all packets necessary to maintain a contiguous streamed file.

Referring now to Fig. 6, in an exemplary process 600 for iterative caching, the SDA receives a request from an end-user for streamed content with a specified bit rate and a starting play position P, step 602. The SDA first checks, step 604, if the requested stream set at the specified transmission bit rate and play position P is already in memory. If it is determined in step 604 that no part of the stream is in the cache, then a new file is allocated in the cache, for example, based on information in the file header. Conversely, if the requested set is located in memory, the play position P is set according to the user's request, step 608, and the cache fill initiate horizon (CFIH) and the cache fill terminate horizon (CFTH) are both set based on the play position P and assumed viewing preference of the user, step 610. If it is determined in step 612 that not all packets for streaming the requested file are present within the CFIH, a process for filling the cache up to the CFIH is initiated, step 614. In step 616 it is checked if the packet is present at the play position P. If the packet is present, the SDA begins to stream the file to the user, step 620, otherwise the process 600 waits for the packet to arrive, step 618. After the packet at the play position P is sent to the user, both the play position P and the CFIH are advanced by one packet, step 622, whereafter the process 600 returns to step 612. As discussed above, only those packets are cached that are not already present in the cache.

Fig. 7 illustrates the cache fill process 700 for filling the cache up to the CFTH. In response to a request from a user for streaming content, packets are sent to the SDA cache, step 702. If it is determined in step 704 that the end-of-stream (EOF) is reached or the user has terminated the streaming request, then the connection to the content server is severed, step 710. Otherwise, it is checked in step 706 if all packets for the requested stream have been cached up to the CFTH. If not all packets have been cached, a missing packet is added to the cache and the CFTH is advanced by one packet, step 709, with the process 700 returning to step 704. Conversely, if all packets up to the CFTH have been cached, the process 700 checks if the CFIH has advanced and increments the CFTH to create a "sliding window" with (CFTH – CFIH = constant) to keep an anticipated number of packets (for example, 30 seconds of streamed content) in the cache, step 709. The

process 700 then continues to step 708. Otherwise, the process 700 goes to step 710 and the connection to the content server is severed as before.

The process of iterative caching described above provides an efficient means for provisioning content to be streamed to an end-user with a predictable acceptable quality, as expressed by the Quality metric. Since an SDA is designed to potentially handle large numbers of clients, in particular large numbers of concurrent real-time multimedia streams, the SDA's cache needs to be optimized for its particular resource utilization patterns, which in turn are highly dependent upon the type of content being requested and the Quality value that is desired. There are a variety of file system allocation and more particularly buffer-cache optimizations that can be used to enhance the performance of SDA's.

Referring now briefly to Fig. 10, a file can also be recached by the process described above with reference to Fig. 5 if a file has been partially evicted from the cache, and is to be stored, for example, due to renewed user interest. The exemplary file portion 104c of file 104 can be recached from the content provider and a longer file portion resembling the portion 104a can be stored in the cache memory of the SDA. Likewise, missing content from files 102 and 106 can be recached by the same process. Accordingly, iterative caching is complementary to cach eviction, which will be described later.

Referring now to Fig. 8, in a process 800 for managing cache content, data sets in a stream that does not represent a complete stream and may therefore not be usable for streaming to a user, may still be kept in the cache. For example, it may be beneficial to continue to cache streams from a content server that are likely to be used by another user after the present user has disconnected from the SDA. The process 800 of Fig. 8 begin in an idle state 802, with a user starting to receive streamed content, step 804. The process 800 checks in step 806 if streaming has been interrupted, for example, intentionally by the user or by another service interruption; if not, then step 808 checks if streaming of the file is complete, in which case the file can be left in the cache (at least temporarily, as will be discussed later), step 810. If streaming is not complete, as determined in step 808,

then the process will return to step 804 and streaming of the file will continue. If step 806 detects that file streaming has been interrupted, the it is checked in step 816 if more than a predetermined percentage of the streamed file has been played. If less than the predetermined percentage of the streamed file was played, the cached file may not be useful for future users and will be deleted the file from the cache, step 820. If more than the predetermined percentage has been played, as determined in step 816, for example more than 50% of the total length of the file, and the stream during cache fill meets other requirements, such as the quality metric, then the SDA can continue to cache and store the stream, even though the original client is no longer interested in the stream, steps 818 and 820. This process can therefore advantageously use streams that would otherwise be discarded for streaming to other users, even when the original user did not download the entire file.

In another aspect of efficient cache management, in particular with respect to improved buffer cache ejection, a distinction is made between content (streaming content; low priority) and system data (metadata, applications, configuration files, etc.; high priority).

Bulk content represents the actual data in the multimedia files, while system data represents the metadata about the multimedia files, as well as possibly programs and data used to serve the bulk content. The system data, while representing only a small fraction of the actual content stored on the physical media and used while streaming, tends to be accessed frequently. If bulk data and system data were treated equally by the cache, system data could be emptied prematurely from the cache due to lack of memory. A subsequent failure to access system data in the buffer-cache would then prevent access to the bulk data, degrading the overall performance of the system. Accordingly, the SDA indicates to the buffer cache subsystem which data is bulk data and which is system data, and the buffer cache ejection policies of the system favor keeping system data over bulk data. The resulting reduction in buffer cache misses for system data more than compensates for the increase in retained system data. The overall system

performance increases significantly due to the reduction in disk access attempts to retrieve system data that have been deleted from the buffer cache.

Efficient management of cache content also relates to limiting the amount of data stored in the cache. Data to be streamed are typically delivered from disk storage rather than from a main memory. If a file is not stored on the disk in sequential order, each disk read requiring a disk seek to locate a block of data on the disk before transmitting the next block of data. Disk seeks are time-consuming operations and increase the total disk transfer time. Without appropriate buffering of the streamed content, most streaming applications tend therefore to be governed by the seek performance of the disk storage system. The data from a file may be requested by different users with different streaming rates. Hence, a different number of bits per unit time may be requested with different storage requests. The SDA of invention hence is able to vary the size of each request based upon the bit rate of the stream being served to the client. Once the storage request has been satisfied, the data associated with the request is kept in main memory until consumed by the network connection delivering the streaming data. Varying the size of the storage request allows a trade off between expensive storage requests and the amount of main memory required. Larger individual requests require fewer operations, but consume larger amounts of memory.

Referring now back to Fig. 4, consistent and uninterrupted data delivery can be further improved by double-buffering the data read from disk 31. With double buffering, a second buffer 34 is reading data from disk 31 while the first buffer 33 is streaming the data to the client 12. However, the second buffer 34 is only allocated and reads in from disk 31 after a fixed percentage of the first buffer 33 has been consumed. Disadvantageously, double buffering tends to require more buffer cache per stream than single buffering. This situation can be alleviated to some extent by timing the allocation of buffer space in the second buffer 34 based on the estimated transmit times of the data in the first buffer 33 that have not yet been transmitted. According to this optimization, the second buffer 34 is allocated and a read from disk storage 31 into the second buffer 34 is initiated when the estimated transmission time for the amount of data left in the first buffer 33 (based

on the transmission rate) is approximately equal to the time required to read data from disk 31 into the second buffer 34. With this approach, the second buffer 34 will just become ready for transmission when the first buffer 33 empties.

As mentioned above, the content file received by the SDA from the content provider can represent a superset of the file data packets requested by a client. This superset may contain multiple video streams and hence be quite large and of little use for individual clients. Due to their large size, these files are typically not kept in the SDA's memory, since they can generally be downloaded again from the content provider if the SDA receives additional requests for the file.

Instead, the SDA may "shred" the superset received from the content server into a contiguous client-specific data file for streaming to the client. In addition, the SDA may in the shredding operation assemble from the superset other contiguous files, for example, files with a different streaming bit rate. The captured stream as well as the other files can be evicted from the SDA's memory, for example, based on their frequency of use or other criteria.

Each of the "shredded" data files contains an individual component stream with typically an interleaved audio/video stream of appropriate bit rate to be transmitted to a user. The SDA can dynamically interleave the component streams at the time the data files are placed on the storage medium of the SDA, which reduces processing delays on playback. The process of creating streams pre-processed for later playback is called Stream Shredding. Stream Shredding may be performed either statically before a client request has been issued, or dynamically at the time of a user request. Static Stream Shredding is initiated on the SDA by an administrator request to pre-populate streams on the SDA. This request will cause the creation of data files representing one, several or all possible combinations of the component streams. Stream shredding can be performed when the first user requests a stream combination that does not already exist on the SDA. At the time the stream is collected and shredded for delivery to the client, it is also saved on the storage medium for subsequent use. This shredded stream is then used for all subsequent client requests for the same stream combination. As a result, each

shredded stream file advantageous contains essentially only those data required by a particular common session, with the client receiving a subset of the original data, differentiated, for example, by available bandwidth as before, language preference, or other static or dynamic characteristic of that particular session. This optimization can result in significant performance gains, at a tolerable cost of redundant storage.

As illustrated in Fig. 9, in an exemplary shredding process, the SDA receives a source content file 910 from a content provider and "shreds" the source content file 910 into a number of exemplary contiguous files 920, 930, 940 that are available for streaming to end-users and have different file characteristics, such as different bit rates, different language audio tracks, different video resolution and the like. The original exemplary content file 910 can have a stream header 914 and presentation units 912 containing different exemplary packets 1, 2, 3, and 4. As seen from Fig. 9, for the particular piece of content, the streamed files 920, 930, 940 represent contiguous subsets of the content file 910. The streamed files 920, 930, 940 can also include stream headers 924, 934, 944 representing, for example, different network protocols for connection to the end users, and respective presentation units 922, 932, 942 with network headers 926, 936, 946 and payload data packets 1, 2, 3, 4. These subsets can be rated, as mentioned above, in terms of their streaming characteristic, in particular their streaming bit rate.

The respective presentation units 922, 932, 942 and/or payload data packets 1, 2, 3, 4 of the streamed files 920, 930, 940 are typically arranged in the cache in a particular order which reflects the transmission order to the client. One particular transmission order could be a time-sequential arrangement.

Caches are of finite size and the number of potentially cacheable objects is typically orders of magnitude larger then the cache size. Processes that can more effectively manage the cache space translate directly into operational benefits. For example, the cache should advantageously be able to evict content from the cache to make room for new content that needs to be cached. Therefore, one cache operation is the removal of less frequently accessed items (or files) from the cache

space. For example, the popularity of videos has been shown to follow a Zapf-distribution with a skew factor of 0.271, which means most of the demand (80%) is for a few quite popular video clips. To quantify this result for the SDA, the SDA tracks and controls information for each file served by the SDA, for example, file attributes such as the streaming protocol used (e.g., MMSU, RTSP, HTTP, and the like), which streams within a file are selected, streaming bit rates, stream characteristics (e.g., audio, video, etc.); and length of streams. Recording the attributes enables the illustrative SDA to develop a better picture of what clients are likely to request in future operations. For example, the SDA can record how often a 100 Kb/s stream is selected versus other bit-rates. With this information, the SDA can decide which streams to remove from the cache.

Referring now to Fig. 10, a number of files 102, 104, 106 were shredded from a content file. Each file 102, 104, 106 is adapted for a specific bit rate (n = 1, 2, 3) and characterized by a "hit rate" which is updated periodically. Initially, an entire file may have been cached. After a period of time, content of the cache is purged to make room in the cache. According to an embodiment of the invention, however, instead of the entire streamed file, only a portion of a file 102, 104, 106 is evicted from the cache. As depicted in Fig. 10, file portion 102a of file 102, file portion 104a of file 104, and file portions 106a, 106b of file 106 remain in the cache. After a certain time period has passed with little interest in the file 104 past the beginning portion of the file, the intermediate portion will also be purged from the cache, with only the beginning portion 104c remaining in the cache for future streaming. The criteria used by the SDA for cache eviction will now be described.

For example, the SDA can employ a "least frequently used" algorithm. Thus, if clients request 100 Kb/s streams more often than streams with other streaming rates, then the 100 Kb/s media streams will tend to remain in the SDA longer. In this fashion, the illustrative SDA tends to accumulate media streams that more closely resemble the types of requests that have been seen before, and thus are more likely to be seen in the future.

Alternatively or in addition, the SDA can employ a "least recently used" or "age-based" algorithm for purging outdated media streams from the cache which are expected to be used less and less frequently. The SDA may also define an age horizon beyond which all cached items are deemed to have the same age. For items beyond the horizon, but also for items within the horizon, the SDA may employ the "least recently used" algorithm to make a decision as to which items to purge. The cache may also evict from the least requested streams first those files that have the lowest streaming rates.

Cache retention can also be adapted to the expected viewer habits. For example, shorter files that are more likely to be streamed to a user, can remain in the cache longer than longer files. Also, as described above with reference to file 204, the beginning portion of a file can remain in the cache longer than the middle or end portion of the file since many users may only be interested in a "snapshot" of the file and will play the streamed file for a short duration from the beginning. If necessary, the content removed from the cache can be recached from the content provider. The cache eviction process hence treats each shredded file in the cache as a separately manageable and evictable entity. Moreover, partial components of files and shredded files can be evicted, leaving more popular segments of the files in the cache.

It will be understood that the data sets in all embodiments described herein can be composed of video and audio, text, html files, wherein these data can be combined in the transmitted packets to ensure synchronization.

A barrier to achieving high throughput is the high cost of copying data in the SDA relative to the cost of processing that data. The basic shredding operation described above would require copying the actual data for each subset from the original stream. Therefore, in order to exploit the throughput potential of the SDA, the number of copies between content provider and user must be kept to a minimum. One opportunity to improve performance is to eliminate copying by performing a lookup to locate the cached data and to provide addressability to the cached data, for example, by providing a pointer to the data. This requires

mapping the cached data into host address space. Protocol processing may be performed and protocol headers inserted before the cache is instructed to transmit the packet. This approach is particularly suited for continuous media applications, such as streaming, which involve the transfer of data between the SDA and one or more users without requiring the manipulation of that data. This improves throughput and efficiency of the SDA.

A memory-mapped interface is required to make copy elimination possible. The zero-copy feature has a direct impact on cache performance. A feature of the zero-copy model presented here is that network data is not brought into the cache unless and until it is explicitly copied by the processor, which provides a number of benefits. Firstly, the level of cache residency seen by the rest of the system increases if network data does not enter the cache. Secondly, incoming network data is only brought into the cache if and when the application consumes the data (i.e. as late as possible). This maximizes cache residency by eliminating the potential for context switches between the data being brought into the cache (by the network subsystem) and the data being consumed by the application. Moreover, the performance penalty incurred by making non-cacheable accesses to memory is reduced with protocols that touch only part of a packet (e.g. the header) rather than the entire packet. Such protocols generally sacrifice error detection (by eliminating the checksum, for example). However, the SDA of the invention pre-computes checksums and stores these checksums, as described below. Since in "zero-copy" mode the data remain unchanged, the checksums also remain valid and do not have to be recomputed.

To further increase the streaming efficiency, the SDA can precompute correction information, such checksums of packets of payload data, when the data are cached, or alternatively can use the checksums transmitted by the content provider with the content file, and store the checksums in memory. The checksums relate to the canonical form of the content stored in the SDA and therefore typically does not include packet header, addresses, etc. In other words, payload data packets that are identical except for the header have identical checksums. With this approach, there is no need to recompute the checksums when streaming

the file to the end user. In this way, the SDA of the invention avoids the delay associated with recalculating the checksums each time the SDA plays back the stream and can use the advantageous features associated by "zero-copy" and "zero-touch" transfer of the streamed data through the SDA. Since each protocol supported by the SDA for transmitting content to/from the SDA is able to convert to/from the canonical form stored in the SDA, checksums computed for different streaming protocols, such as TCP, UDP and other proprietary streaming network protocols, can later be used with other protocols when streaming from the cache. The checksums used by TCP, UDP and many streaming protocols can therefore be easily updated on the fly to reflect partial updates only to the data associated with a respective checksum. Typically, data packets for streaming content are already broken into packet-sized units so the check sums of entire packet-sized units of data may be precomputed when the streaming data is written to storage.

The checksum for each fragment can be maintained independently, so the server can reuse fragments without recomputing the checksum of each fragment. This allows portions of a response to be cached and check-summed independently. When constructing a response to a user request, the server only needs to pull together previously cached portions and add the corresponding checksums. The zero-copy feature can also eliminate additional copying into the cache, as described above.

Since typically several clients can request the same content using identical streaming protocols, the transmitted pre-computed checksums are an efficient means for detecting, and optionally correcting, corrupted packets at the end-user site. However, the network protocols used between the SDA and content provider may or may not compensate for lost or corrupted data. For performance and scalability reasons, although not all errors are corrected, errors are typically detected.

Another aspect of efficient management of cache content relates to the efficient streaming of content independent of the streaming protocol. Different protocols can be used to deliver the same piece of content. Some protocols are

optimized use with a local area network (LAN) while other protocols are optimized for delivery through firewalls. Once content is cached, protocols must be used to authenticate access to the content and to send usage information about what content has been delivered. Traditionally, caches have tied the protocol a client used to request content to the protocol the cache used to retrieve, authenticate and log access to the content.

Protocol-independent caching is a process whereby the protocol used by a client to access content from cache is separated from the protocol used to fetch the content from the content provider into the cache. This involves translating content as well as authentication and usage information from a protocol-specific form into a protocol-independent (canonical) form when content is acquired; and then translating the protocol independent-form back into a protocol-dependent form when a user makes a request to the cache from streaming. For example, in one embodiment, Windows Media Format received via the MMST, SSH/SCP and FTP protocols can thereby be streamed, authenticated and logged to clients using MMSU, MMST, or MMSH protocols. Conversely, Real Media® content received via HTTP, SSH/SCP and FTP protocols can then be streamed, authenticated and logged to clients using different RTSP/RTP/RDT and PNA protocols.

The servers of content providers as well as the SDA, as mentioned above, typically include storage subsystems having persistent memory, such as disk and/or tape drives, and short-term memory, such as RAM, for storing the content that will be served and/or replicated. Efficient handling of client requests for streaming multimedia files requires not only an optimized cache, but also an optimization of the storage subsystem and, more particularly, the data structure of these files.

Conventional file systems perform disk-block allocation using fixed-size allocation units, regardless of the type of content being stored in the file. These allocation units tend to be relatively small, often on the order of 4 KB, which is adequate for many computer application files and data files. However, these allocation units are significantly smaller than the size of multimedia files providing streamed content. Accordingly, finding the correct blocks of a multimedia file

could require a large number of disk "seek" operations, which can reduce throughput and degrade disk performance. Optimizing the allocation units for multimedia files can therefore be expected to result in substantial performance gains.

Streaming applications can be written to read largely deterministic sized portions as they are being streamed. An optimum size of the allocation units can be determined by analyzing the bit rate of the streamed files being stored on the disk. Optimizing the storage subsystem to allocate space in this manner eliminates or at least substantially reduces any intra-file-read seeks, while avoiding the storage inefficiencies of storing all files contiguously.

The allocation units for multimedia files can be optimized, for example, by dynamically building variable size allocation units so that the streamed files can be read at the same disk request frequency (e.g., number of seeks per second), regardless of the bit-rate of the stream. It will be understood that due to potential non-linear characteristics of the memory subsystems (such as virtual or physical page size, map region allocation performance, etc.) and for ease of implementation, there may be a range of variable size allocation units for various bit-rates. However, the ability to read large portions of files adapted for higher bit-rates and having larger disk allocation sizes can still improve disk performance even if this approach requires increased memory storage for the read-ahead portion of low bit-rate streams. File allocation management can be conventional and can include a storage metadata layout, frequently also referred to as file allocation table.

To accommodate variable size allocation units, file allocations can be made in a cascading fashion wherein as the file size grows, the allocation size grows as well. This can be accomplished by organizing the metadata structure in form of an inode. An inode is a data structure holding information about files. There is an inode for each file and a file is uniquely identified by the file system on which it resides and its inode number on that system. The inode structure provides embedded pointers to data blocks, and a pointer to an indirect block, which itself can contain more pointers to data blocks of different size, and possibly a pointer to

a double-indirect block, which once more can contain pointers to more indirect blocks, and so on.

Referring now to Fig. 11, in an embodiment particularly suited for streaming applications, files are allocated in a cascading fashion wherein the allocation size can increase with bit rate. For example, the allocation units 112 (indicating a direct block), 113 (indicating an indirect block), and 114 (indicating a double-indirect block) can be laid out on a disk storage medium, each with a conventional (small) block size. However, whereas some of the blocks can be direct blocks 112 that include conventional small data blocks 115, which may be suitable for low bit rate streaming, for example, for streaming to a 56 kB modem, other blocks can be indirect blocks 113, each of which can in turn point to direct blocks 112' containing larger data blocks 116 adapted for streaming at a higher bit rate. Likewise, double indirect blocks 114 can each point to differently sized indirect blocks 117 which each include pointers to corresponding direct blocks 112", 112"' containing again larger data blocks 116 adapted for streaming at an even higher bit rate. The large data blocks 116 addressed by the indirect blocks can all be contiguous. Alternatively, the double-indirect blocks 114 can point directly to extra-large data blocks (not shown), in the same manner as the indirect blocks 113 point to the large data blocks, since the start and length of the large and extra-large data block is the only information needed for streaming. In either of these schemes, one may predefine the size of the small and large (or extra-large) data blocks, as well as the number of pointers, to optimize the allocation patterns for files depending on various bit rates.

The aforedescribed allocation scheme can also optimize the storage-efficiency/ performance balance for files stored on the SDA, which includes small files (e.g. streaming content meta-information) and large files (e.g. streaming content data).

While the invention has been disclosed in connection with the preferred embodiments shown and described in detail, various modifications and improvements thereon will become readily apparent to those skilled in the art.

Accordingly, the spirit and scope of the present invention is to be limited only by the following claims.

CLAIMS

1. Method of producing a cache file, comprising:

receiving a content file from a content provider;

defining a quality threshold value, said quality threshold value representing an number of packets representative of the content file;

defining a quality metric of the received content file, said quality metric representing a number of received packets out of the number of packets representative of the content file;

testing the quality metric of the received content against a first quality threshold value to determine if the quality metric is equal to the a first quality threshold value; and

if the quality metric is equal to the first quality threshold value caching the received content from a content provider and producing a cache file; and

if the quality metric is smaller than the first quality threshold value discarding the cached content.

2. The method of claim 1, wherein producing the cache file includes assembling the packets of the cached content as a contiguous cache file having a predetermined order of packets, and storing the contiguous cache file.

3. The method of claim 1, wherein testing the quality metric includes comparing the received content having a second quality metric with a second quality threshold value and transmitting the received content to the client if the second quality metric is greater than second quality threshold value, and continue caching content from the content provider until the first quality metric is greater than first quality threshold value.

4. The method of claim 1, wherein the first and second quality threshold values represent the overall number of packets representative to the content.

5. The method of claim 1, wherein the first and second quality threshold values are identical.

6. The method of claim 1, wherein storing the cache file further comprises

requesting from the content provider packets that are missing from the content;

receiving the requested packets;

updating the stored content with the received packets until the first quality metric of the updated cache file is equal to the first quality threshold value.

7. The method of claim 1, wherein defining a quality threshold value further includes determining a required transmission order of the cached packets and testing the quality metric includes determining an actual order of the received packets.

8. The method of claim 7, wherein the transmission order is a time-sequential order.

9. The method of claim 1, wherein producing the cache file includes shredding the source content file into at least one shredded cache file having a streaming characteristic, and storing the at least one shredded cache file.

10. The method of claim 9, wherein the streaming characteristic is a streaming bit rate to a user.

11. The method of claim 1, further comprising receiving or computing a checksum for the received content and storing said checksum.

12. The method of claim 11, further including associating said checksum with the cache file and transmitting with the cache file when streaming the cache file to a user.

13. The method of claim 1, wherein producing the cache file includes removing network protocol information from the cache file and storing the cache file as a canonical file.

14. The method of claim 11, wherein the cache file is transmitted to a user via the TCP or UDP network protocol and the checksum is TCP or UDP compatible.

15. The method of claim 3, wherein transmitting the cached content to the client includes multicasting said cache file to a plurality of clients and transmitting said checksum to the plurality of clients.

16. The method of claim 15, wherein the checksums are compatible with the MMS multicasting protocol.

17. Method of producing a cache file, comprising:

receiving a content file from a content provider;

defining a quality threshold value of the content file, said threshold value representing a number of packets representative of the content file;

defining a quality metric of the received content, said quality metric representing a number of the received packets out of the number of packets representative of the content file;

caching content from the content provider if the quality metric of the cached content equal to the quality threshold value;

streaming the cached content to a user; and if the user interrupts streaming,

continuing caching a remaining portion of the content if the cached content represents at least a predetermined fraction of the content file; and

deleting the cached content if the cached content represents less than the predetermined fraction of the content file.

18. Streaming delivery accelerator (SDA) for producing a cache file, the SDA comprising:

at least one input channel that receives a content file from a content provider;

a cache memory that stores content of the received content file; and

at least one output channel with a predetermined bandwidth for streaming a cache file produced from the received content;

wherein the received content has a quality metric, said quality metric representing a number of received packets out of a number of packets representative of the content file, and wherein the received content is stored in cache memory as a cache file if the quality metric of the received content is at least equal to a quality threshold value.

19. The SDA of claim 18, wherein if the quality threshold value is less than a predetermined value, the SDA requests from the content provider packets that are missing from the content, and incrementally updates the cache file with the received missing packets, until the quality metric of the updated cache file is at least equal to the quality threshold value.

20. The SDA of claim 18, wherein the cache file comprises interleaved video and audio data.

21. The SDA of claim 18, wherein the quality threshold value represents a time-ordered reception of the overall number of packets representative of the content file.

22. Method for shredding a content file to create a contiguous cache file for rapid streaming delivery, comprising:

receiving a content file from a content provider;

identifying content file packets to be associated with the cache file;

assembling the identified content file packets into the contiguous cache file having a predetermined streaming characteristic; and

storing the contiguous cache file on a persistent storage medium.

23. The method of claim 22, wherein assembling the identified packets includes generating pointers to the packets of the received content file and assembling the contiguous cache file by fetching from the received content file those packets identified by the pointers.

24. The method of claim 22, wherein assembling the identified packets includes assembling the identified packets in the contiguous cache file as an ordered sequence of the packets.

25. The method of claim 22, wherein the ordered sequence corresponds to a transmission sequence of the packets to a user.

26. The method of claim 22, wherein the ordered sequence corresponds to a time-ordered sequence of the packets

27. The method of claim 22, further including assembling the identified content file packets into a plurality of contiguous cache files having different streaming characteristics and storing said plurality of contiguous cache files in the persistent storage medium.

28. The method of claim 22, wherein assembling includes dynamically generating the contiguous cache file from the content file concurrent with a streaming request received from a user.

29. The method of claim 22, wherein assembling includes transmitting the cache file to a user.

30. The method of claim 22, wherein the streaming characteristic includes a data transmission rate to a user.

31. The method of claim 22, wherein the content file is received from the content provider via a first network protocol, and the streaming characteristic includes transmission of the contiguous file to a user via a second network protocol that is different from the first network protocol.

32. The method of claim 22, wherein the content file is received from the content provider via a first network protocol, and the streaming characteristic includes transmission of the contiguous file to a user via a second network protocol that is identical to the first network protocol.

33. The method of claim 31, wherein creating the contiguous file includes removing information about the first network protocol and storing the contiguous file in canonical form.

34. The method of claim 32, wherein creating the contiguous file includes removing information about the first network protocol and storing the contiguous file in canonical form.

35. The method of claim 31, wherein the first network protocol includes at least one of a MMST, SSH/SCP, HTTP and FTP protocol, and the second network protocol includes at least one of a MMSU, MMST, MMSH, RTSP/RTP/RDT and PNA protocol.

36. The method of claim 22, further comprising associating a checksum with a corresponding identified packet of the contiguous file, wherein said checksum is received with or computed from a packet of the content file.

37. The method of claim 22, wherein the streaming characteristic includes data selected from the group consisting of video, audio and text.

38. Streaming delivery accelerator (SDA) for shredding a content file to create a contiguous cache file for rapid streaming delivery, comprising:

at least one input channel that receives a content file from a content provider;

at least one output channel with a predetermined bandwidth for streaming the contiguous cache file to a user;

a shredder that identifies content file packets to be associated with the contiguous cache file and assembles the identified packets into the contiguous cache file having a predetermined streaming characteristic; and

a persistent storage device that stores the contiguous cache file.

39. The SDA of claim 38, wherein the shredder provides a pointer to an identified content packet and assembles the contiguous cache file by fetching from the received content file the content packets identified by the pointers.

40. The SDA of claim 38, wherein the contiguous cache file is ordered so as to correspond to a transmission order of the packets to the user.

41. The SDA of claim 38, wherein the streaming characteristic includes a data transmission rate to a user.

42. The SDA of claim 38, wherein the at least one input channel operates using a first network protocol, and the at least one output channel operates using a second network protocol that is different from the first network protocol.

43. The SDA of claim 38, wherein the at least one input channel operates using a first network protocol, and the at least one output channel operates using a second network protocol that is identical to the first network protocol.

44. The SDA of claim 38, wherein the contiguous cache file stored in the persistent storage device is stored as a canonical file.

45. The SDA of claim 38, wherein a checksum associated with a payload data packet is stored in the persistent storage device, said checksum received from the content provider or computed from packets of the content file, said checksums being associated with packets of the cache file transmitted to the user.

46. The SDA of claim 45, wherein the checksum of a payload data packet is independent of a transmission protocol used to transmit the packet over the at least one input channel and the at least one output channel.

47. A method of allocating data blocks for streaming a file, the method comprising:

determining a data transfer characteristic for streaming the file;

allocating data blocks having a block size on a persistent storage medium, said block size related to the data transfer characteristic;

storing said file as the data blocks on the persistent storage medium;

transferring the data blocks from the persistent storage medium into a first buffer memory, with the data blocks in the first buffer memory having the block size;

determining an actual bit rate to a user;

allocating a second buffer memory to receive from the persistent storage medium additional data blocks based on the actual bit rate to a user, with the data blocks in the second buffer memory having the block size; and

transmitting the additional data blocks from the persistent storage medium to the second buffer memory when an estimated transmit time for streaming a portion of the data blocks remaining in the first buffer to a user is approximately equal to a time required to transfer the additional data blocks from the persistent storage medium into the second buffer.

48. The method of claim 47, wherein the transfer characteristic is a maximum bit rate for the file.

49. The method of claim 48, wherein allocating data blocks on the persistent storage medium includes

defining data blocks that have at least two block types;

associating a first block type with a direct block comprising the data blocks and associating a second block type with an indirect block comprising a pointer to a plurality of other direct blocks or of additional indirect blocks; and

storing on the persistent storage medium a contiguous sequence of at least the blocks that are addressed by the pointer.

50. The method of claim 49, wherein the block size of the additional direct blocks is greater than the block size of at least one of the indirect blocks and the blocks of the first block type.

51. Device for allocating data blocks for streaming a file, the device comprising:

a persistent storage medium storing the data blocks of the file, said data blocks having a block size allocated related to a data transfer characteristic;

a first buffer memory that receives the data blocks from the persistent storage medium, said data blocks in the first buffer memory having the block size;

a second buffer memory that receives additional data blocks from the persistent storage medium, said data blocks in the second buffer memory having the block size,

wherein the additional data are received in the second buffer memory when an estimated transmit time for streaming a portion of the data blocks remaining in the first buffer to a user is approximately equal to a time required to transfer the additional data blocks from the persistent storage medium into the second buffer.

52. The device of claim 51, wherein the transfer characteristic is a maximum bit rate for the file.

53. The device of claim 51, wherein the persistent storage medium is a disk storage medium.

54. The device of claim 51, wherein the first buffer memory and the second buffer memory are volatile memory devices.

55. The device of claim 54, wherein the volatile memory devices comprise random access memory (RAM).

56. The device of claim 51, wherein the block size includes a plurality of block sizes, said block sizes organized in an inode structure and determined by at least one of a maximum streaming rate of the file and a maximum seek time for the data blocks on the persistent storage medium.

57. Streaming delivery accelerator (SDA) for efficiently streaming a file, comprising:

at least one input channel that receives a content file from a content provider;

at least one output channel with a predetermined streaming characteristic for streaming a cache file to a user;

a persistent storage medium that caches data blocks of the content file and stores the data blocks as the cache file, said data blocks having a block size allocated related to a data transfer characteristic;

a first buffer memory that receives the data blocks from the persistent storage medium, said data blocks in the first buffer memory having the block size;

a second buffer memory that receives additional data blocks from the persistent storage medium, said data blocks in the second buffer memory having the block size,

wherein the additional data are received in the second buffer memory when an estimated transmit time for streaming a portion of the data blocks remaining in the first buffer to the user is approximately equal to a time required to transfer the additional data blocks from the persistent storage medium into the second buffer.

58. The SDA of claim 57, wherein the persistent storage medium is a disk storage medium.

59. The SDA of claim 57, wherein the first buffer memory and the second buffer memory are volatile memory devices.

60. The SDA of claim 57, wherein the streaming characteristic includes a data transmission rate to the user.

61. The SDA of claim 57, wherein the at least one input channel operates using a first network protocol, and the at least one output channel operates using a second network protocol that is different from the first network protocol.

62. The SDA of claim 57, wherein the at least one input channel operates using a first network protocol, and the at least one output channel operates using a second network protocol that is identical to the first network protocol.

63. The SDA of claim 57, wherein the cache file stored in the persistent medium device is stored as a canonical file.

64. The SDA of claim 57, wherein data blocks of the stored cache file include checksum representative of payload data.

65. The SDA of claim 57, wherein the data blocks include a checksum, and wherein the checksum received with the content file is used for computing a checksum of the cache file streamed to the user.

66. Method for allocating cache memory for streaming, comprising:

receiving a request from a user for streaming content from a content file, said request including a play position;

checking if the content is in cache memory, and if the content is not in memory, caching the content at least at the play position from a content provider;

determining a cache fill initiate horizon (CFIH), said CFIH defining a cache memory content sufficient to begin streaming to the user, and

caching content from the content provider that is missing between the play position and the CFIH;

defining a cache fill terminate horizon (CFTH), said CFTH defining additional cache memory content sufficient for maintaining a content stream of predetermined duration after streaming to the user has begun;

advancing said CFIH and CFTH synchronously during play; and

caching content from the content provider that is missing between the CFIH and the CFTH.

67. The method of claim 66, wherein said setting of the CFIH and the CFTH depends on a transmission rate of the streamed content to the user.

68. The method of claim 66, wherein said setting of the CFIH and the CFTH depends on a header duration information of the content file.

69. The method of claim 66, wherein said setting of the CFIH and the CFTH depends on a bit rate of the content file.

70. The method of claim 66, wherein said setting of the CFIH and the CFTH depends on a time for establishing a connection to the content provider.

71. The method of claim 66, further including stopping a position of the CFTH when reaching an end of file and disconnecting from the content provider.

72. The method of claim 66, further including disconnecting from the content provider when the content between the play position and the CFTH is in the cache memory.

73. The method of claim 66, further including:

upon detection of an interruption in streaming the content to the user, determine if the cache memory contains more than a predetermined fraction of the content file; and

if the cache memory contains more than the predetermined fraction of the content file, finish caching the remaining fraction of the content file from the content provider; and

if the cache memory contains less than the predetermined fraction of the content file, purge the cached content from cache memory.

74. Method of managing cache memory for streaming, comprising:

receiving from a content provider content of a content file;

caching said content and storing at least a portion of said content as a cache file in a cache memory for streaming;

collecting a request history for streaming of the cache file;

determining from the request history the portion of the cache file that meets predetermined cache eviction criterion; and

evicting from the cache memory the determined portion of the cache file that meets the cache eviction criterion.

75. The method of claim 74, wherein the cache eviction criterion includes a fixed time period, an elapsed time since a previous request, a frequency of previous requests, and a quality metric.

76. The method of claim 74, wherein the cache eviction criterion includes a location of the cached content within a cache file.

77. The method of claim 76, wherein the location includes at least one of a central and a trailing segment of the cache file.

78. The method of claim 74, further including storing said content in the cache file as payload data and as system data, and evicting from the cache memory the payload data of the determined portion of the cache file before evicting the system data of the determined portions.

79. The method of claim 78, wherein the system data include metadata.

80. A computer-readable medium containing instructions for causing a computer to manage a cache memory for streaming, including:

computer instructions for receiving from a content provider content of a content file;

computer instructions for caching said content and storing at least a portion of said content as a cache file in a cache memory for streaming;

computer instructions for collecting a request history for streaming of the cache file;

computer instructions for determining from the request history the portion of the cache file that meets predetermined cache eviction criterion; and

computer instructions for evicting from the cache memory the determined portion of the cache file that meets the cache eviction criterion.

81. The computer-readable medium of claim 80, wherein the cache eviction criterion includes a fixed time period, an elapsed time since a previous request, a frequency of previous requests, and a quality metric .

82. The computer-readable medium of claim 80, wherein the cache eviction criterion includes a location of the cached content within a cache file.

83. The computer-readable medium of claim 82, wherein the location includes at least one of a central and a trailing segment of the cache file.

84. The computer-readable medium of claim 80, further including storing said content in the cache file as payload data and as system data, and evicting from the cache memory the payload data of the determined portion of the cache file before evicting the system data of the determined portions.

85. The computer-readable medium of claim 84, wherein the system data include metadata.

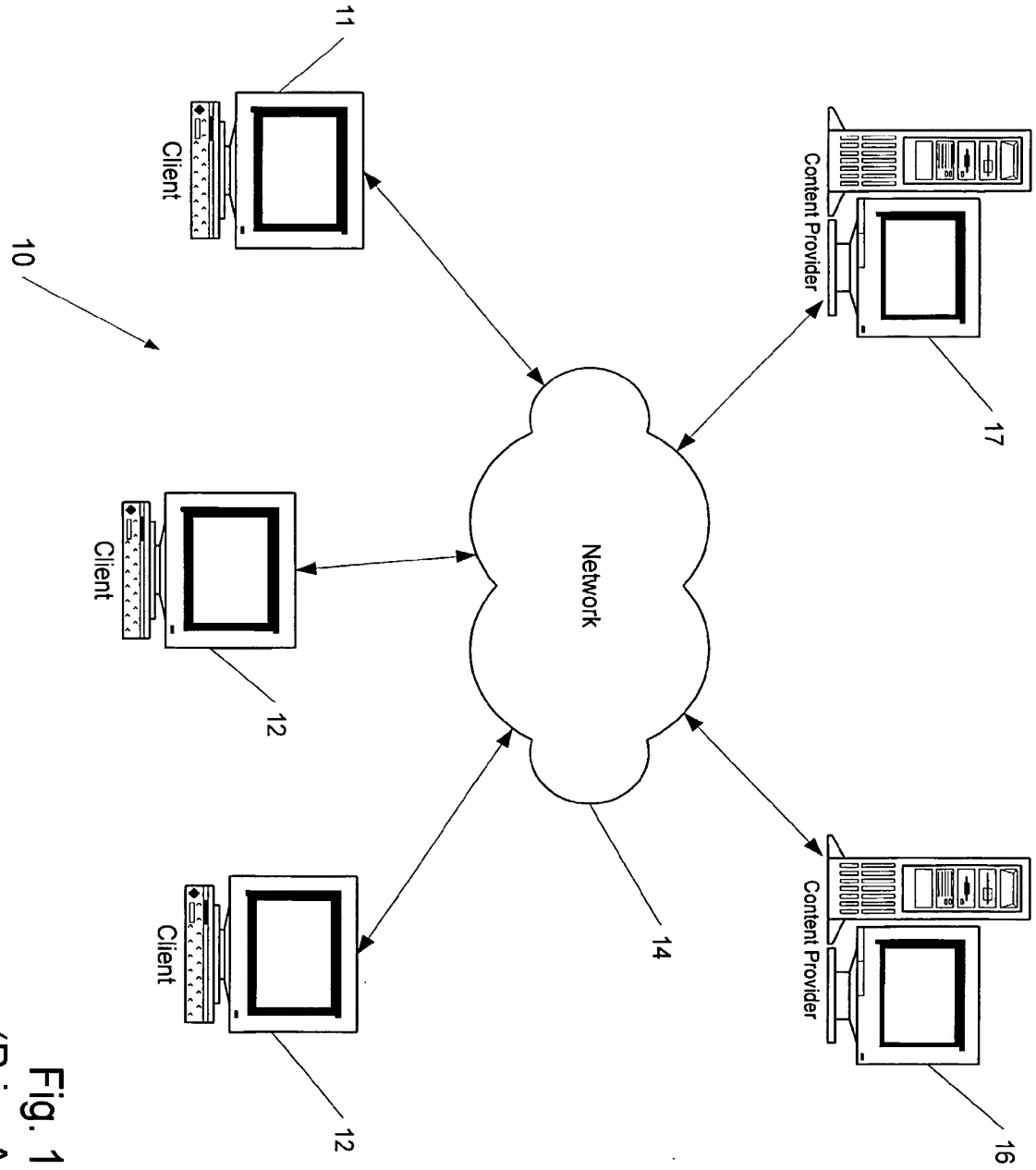


Fig. 1
(Prior Art)

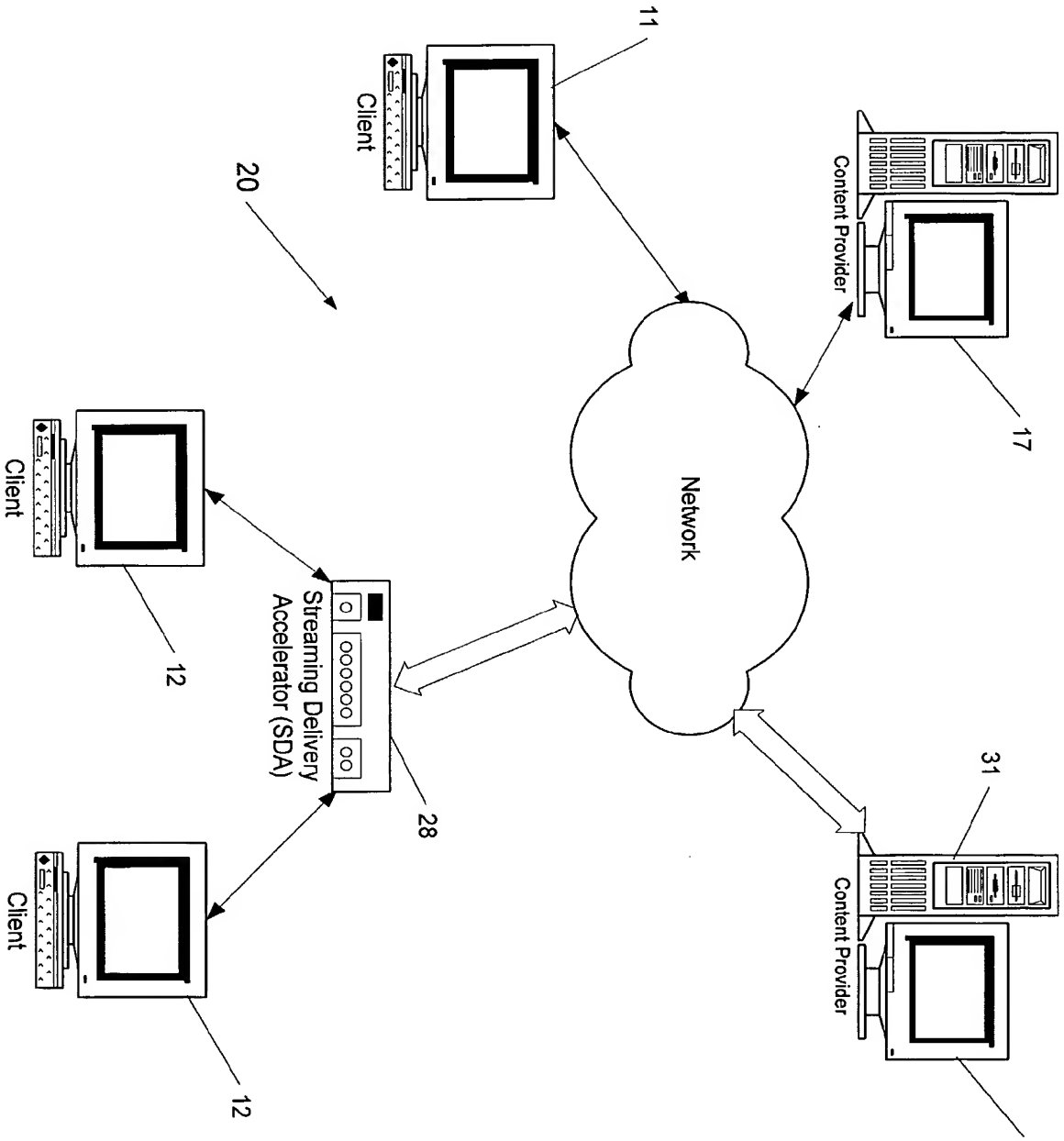

11
Client
20
Content Provider
17
Network
Client
12
Streaming Delivery Accelerator (SDA)
28
31
Content Provider
16
Client
12

Fig. 2

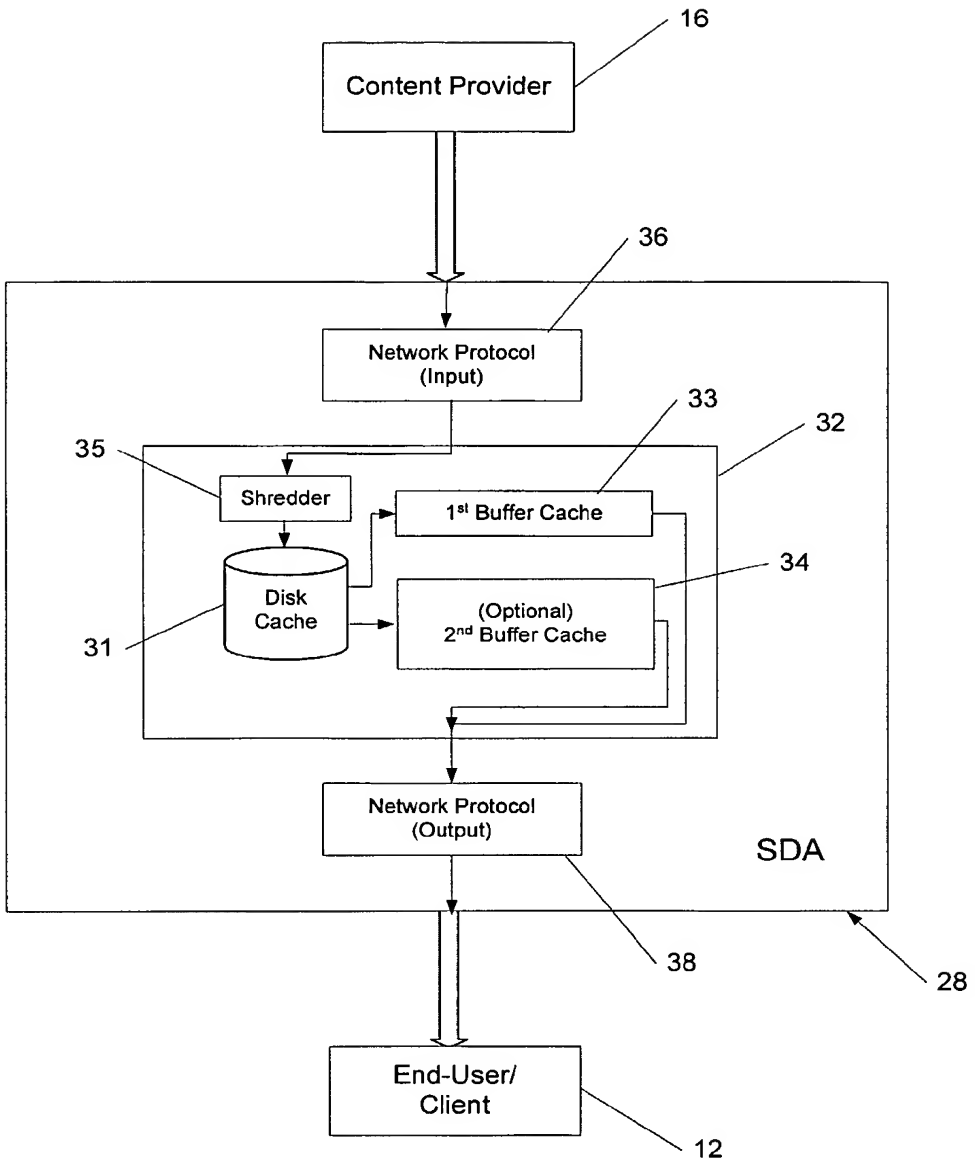

16
Content Provider
36
Network Protocol
(Input)
33
32
35
Shredder
1st Buffer Cache
34
Disk
Cache
(Optional)
2nd Buffer Cache
31
Network Protocol
(Output)
SDA
38
28
End-User/
Client
12

Fig. 3

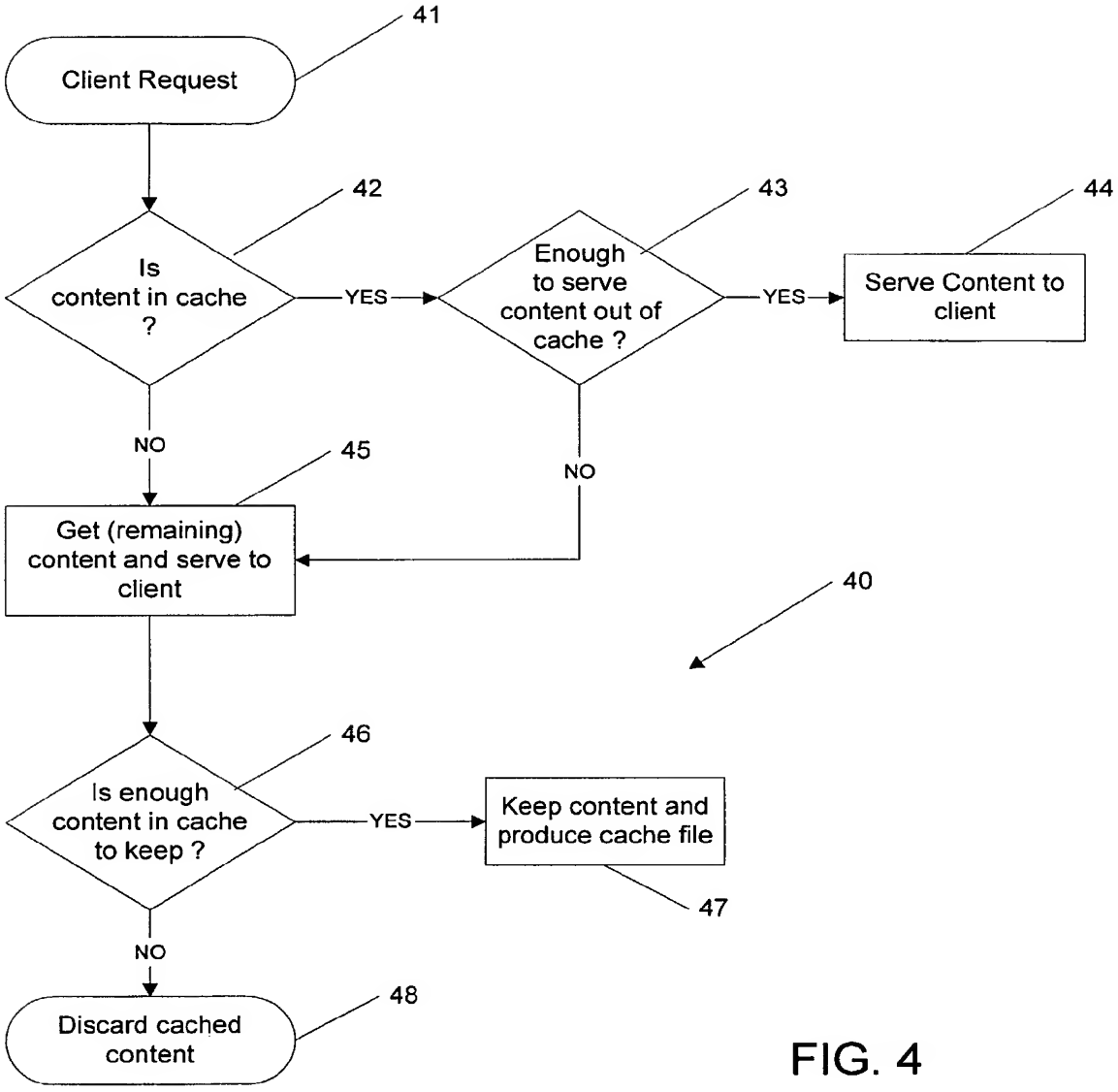

41
Client Request
42
Is content in cache ?
YES
43
Enough to serve content out of cache ?
YES
44
Serve Content to client
NO
45
NO
Get (remaining) content and serve to client
40
46
Is enough content in cache to keep ?
YES
Keep content and produce cache file
47
NO
48
Discard cached content

FIG. 4

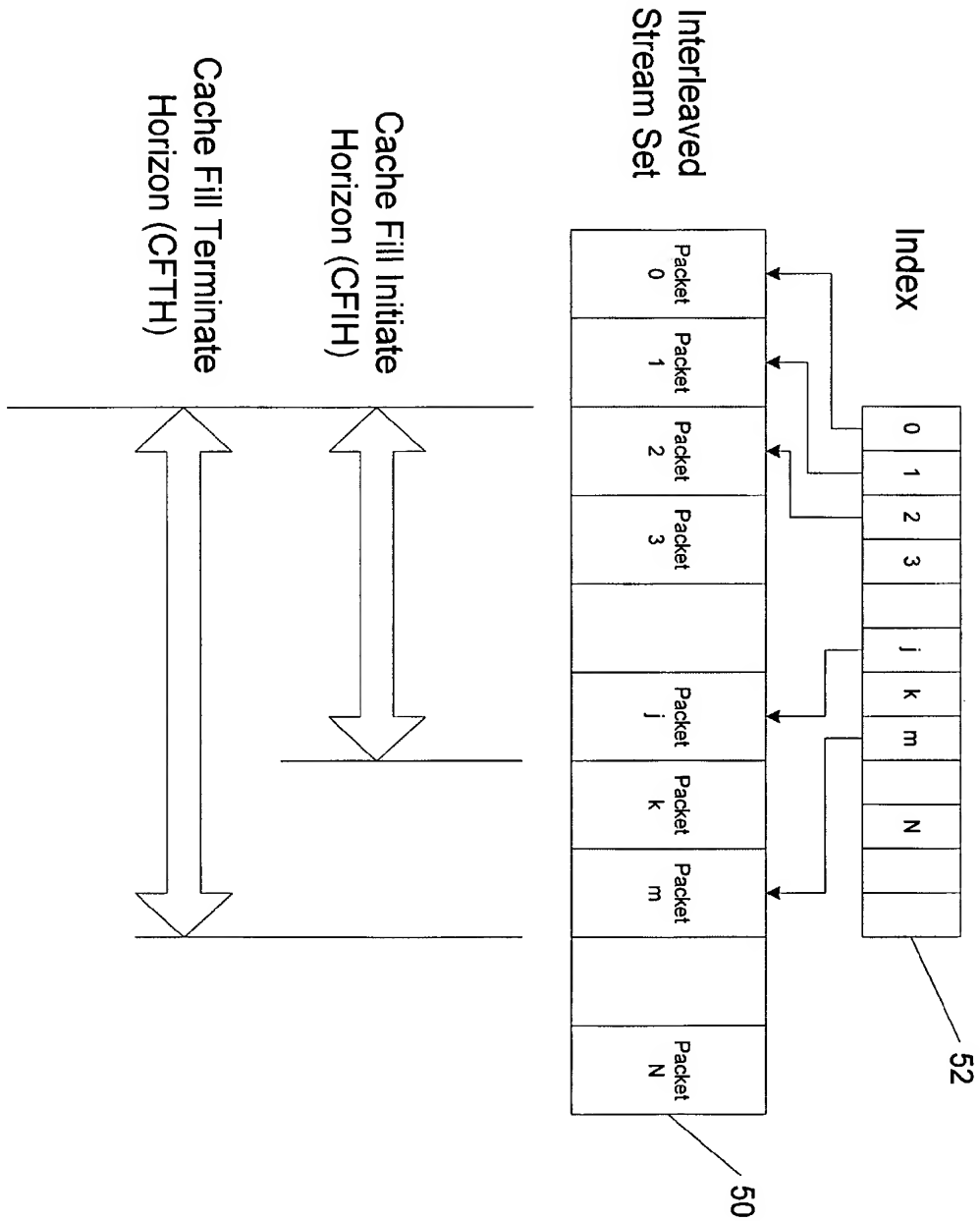

Index
0
1
2
3
j
k
m
N
52
Interleaved Stream Set
Packet 0
Packet 1
Packet 2
Packet 3
Packet j
Packet k
Packet m
Packet N
50
Cache Fill Initiate Horizon (CFIH)
Cache Fill Terminate Horizon (CFTH)
Play Position (P)

Fig. 5

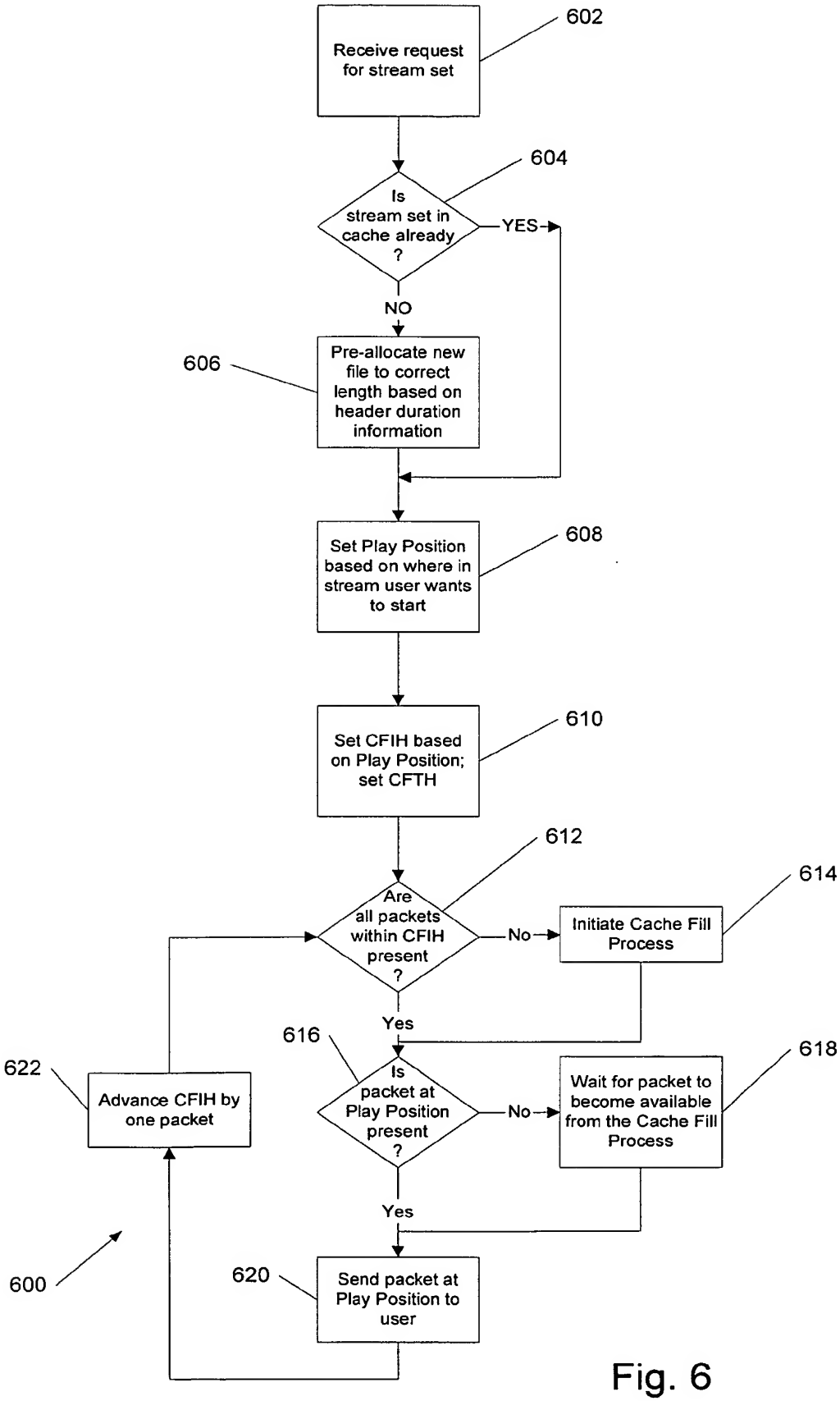

Receive request for stream set
602
Is stream set in cache already ?
604
YES
NO
Pre-allocate new file to correct length based on header duration information
606
Set Play Position based on where in stream user wants to start
608
Set CFIH based on Play Position; set CFTH
610
Are all packets within CFIH present ?
612
No
Initiate Cache Fill Process
614
Yes
Is packet at Play Position present ?
616
No
Wait for packet to become available from the Cache Fill Process
618
Yes
Advance CFIH by one packet
622
Send packet at Play Position to user
620
600

Fig. 6

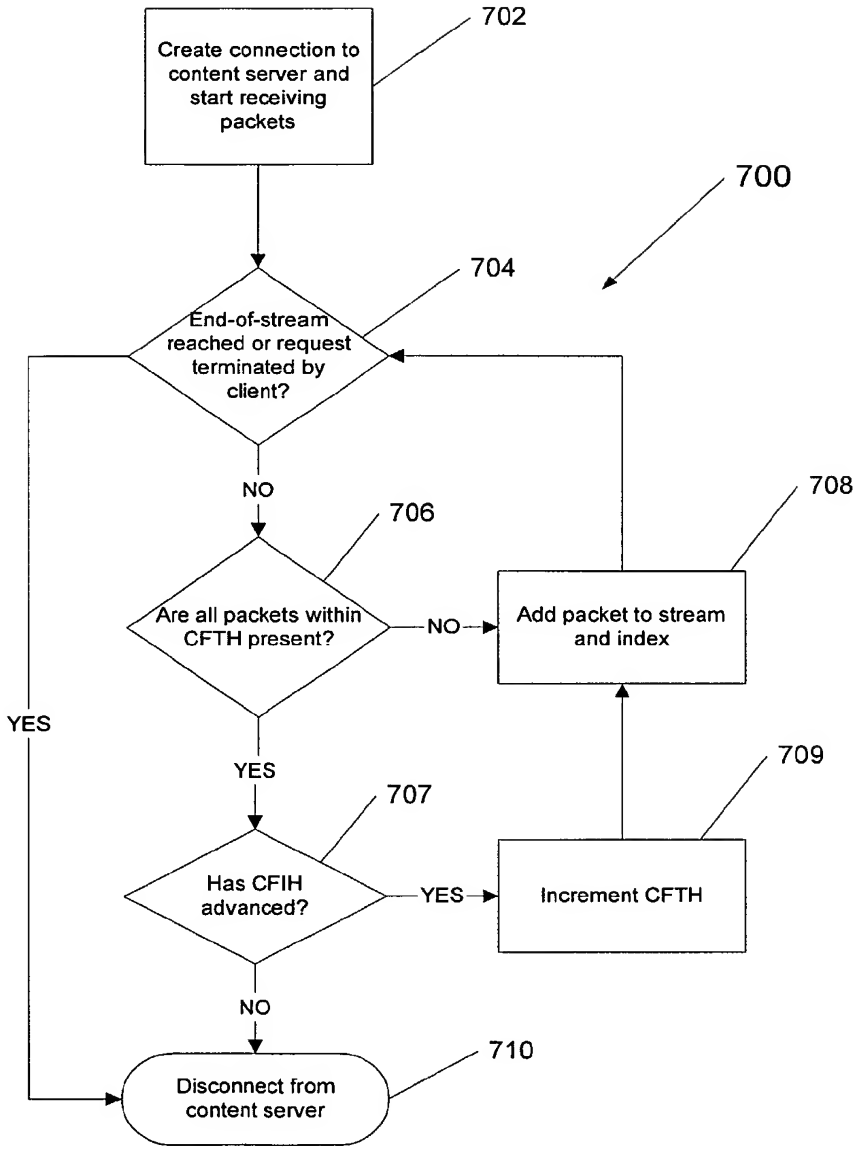

Create connection to content server and start receiving packets
702
700
End-of-stream reached or request terminated by client?
704
NO
YES
Are all packets within CFTH present?
706
NO
Add packet to stream and index
708
YES
Has CFIH advanced?
707
YES
Increment CFTH
709
NO
Disconnect from content server
710

Fig. 7

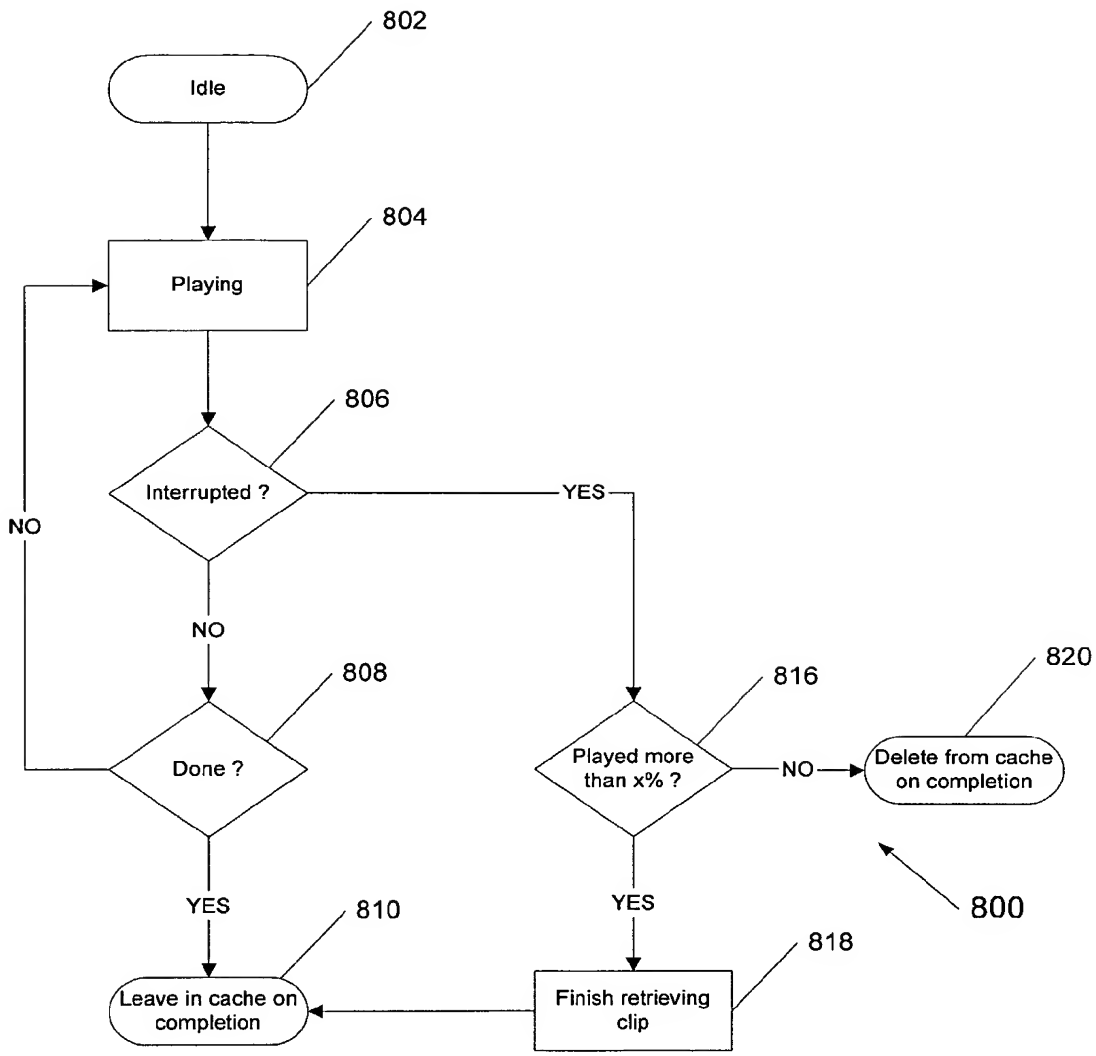

802
Idle
804
Playing
806
Interrupted ?
YES
NO
NO
808
Done ?
YES
810
Leave in cache on completion
816
Played more than x% ?
NO
820
Delete from cache on completion
YES
818
Finish retrieving clip
800

Fig. 8

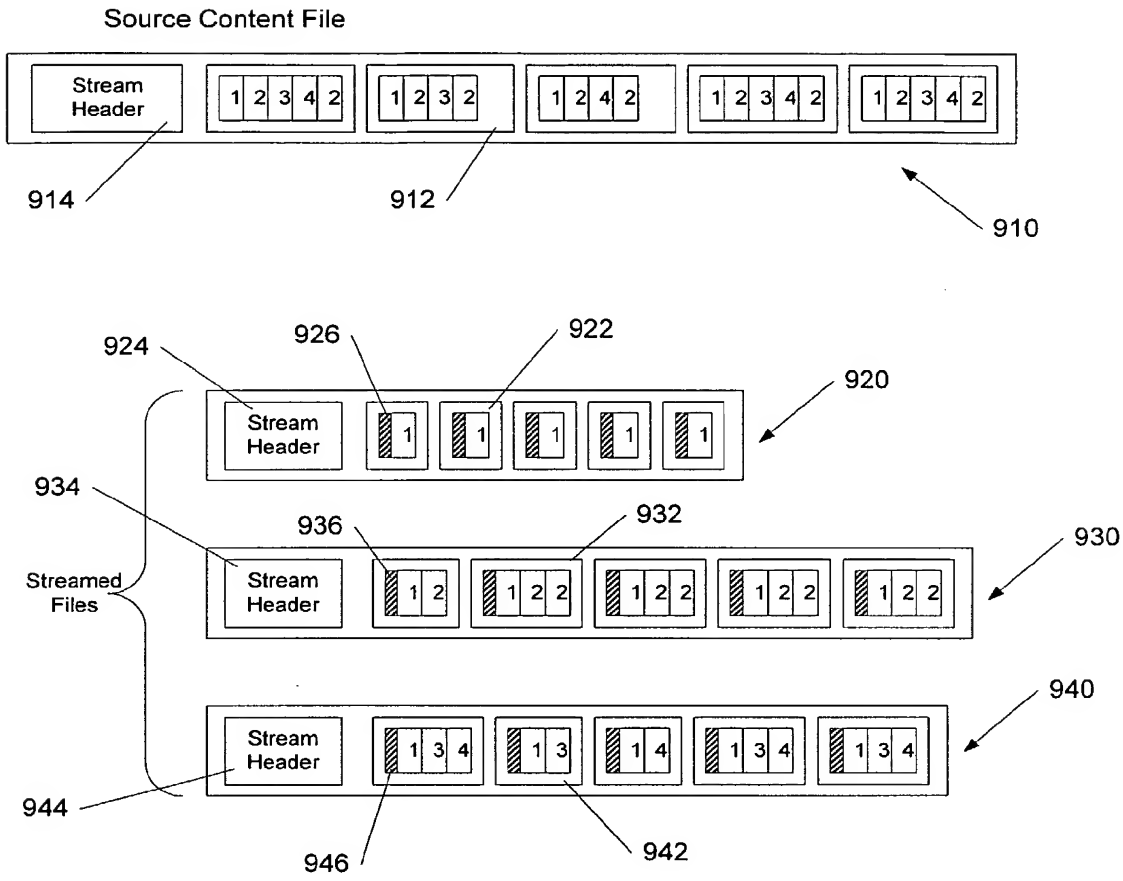

Source Content File
Stream Header
914
912
910
924
926
922
920
934
936
932
930
Streamed Files
940
944
946
942

Fig. 9

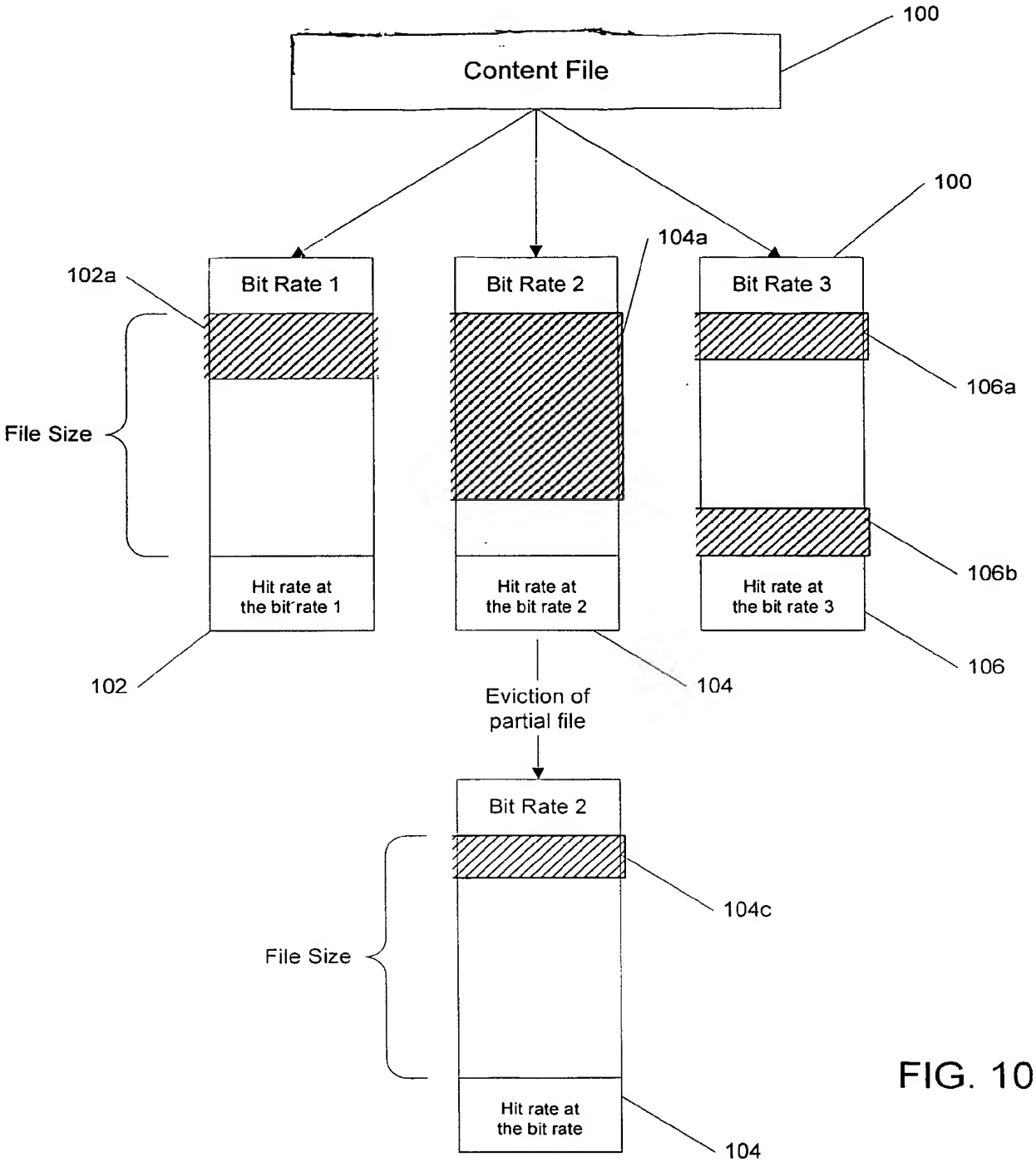

Content File
100
100
104a
102a
Bit Rate 1
Bit Rate 2
Bit Rate 3
106a
File Size
106b
Hit rate at the bit rate 1
Hit rate at the bit rate 2
Hit rate at the bit rate 3
106
102
104
Eviction of partial file
Bit Rate 2
104c
File Size
Hit rate at the bit rate
104

FIG. 10

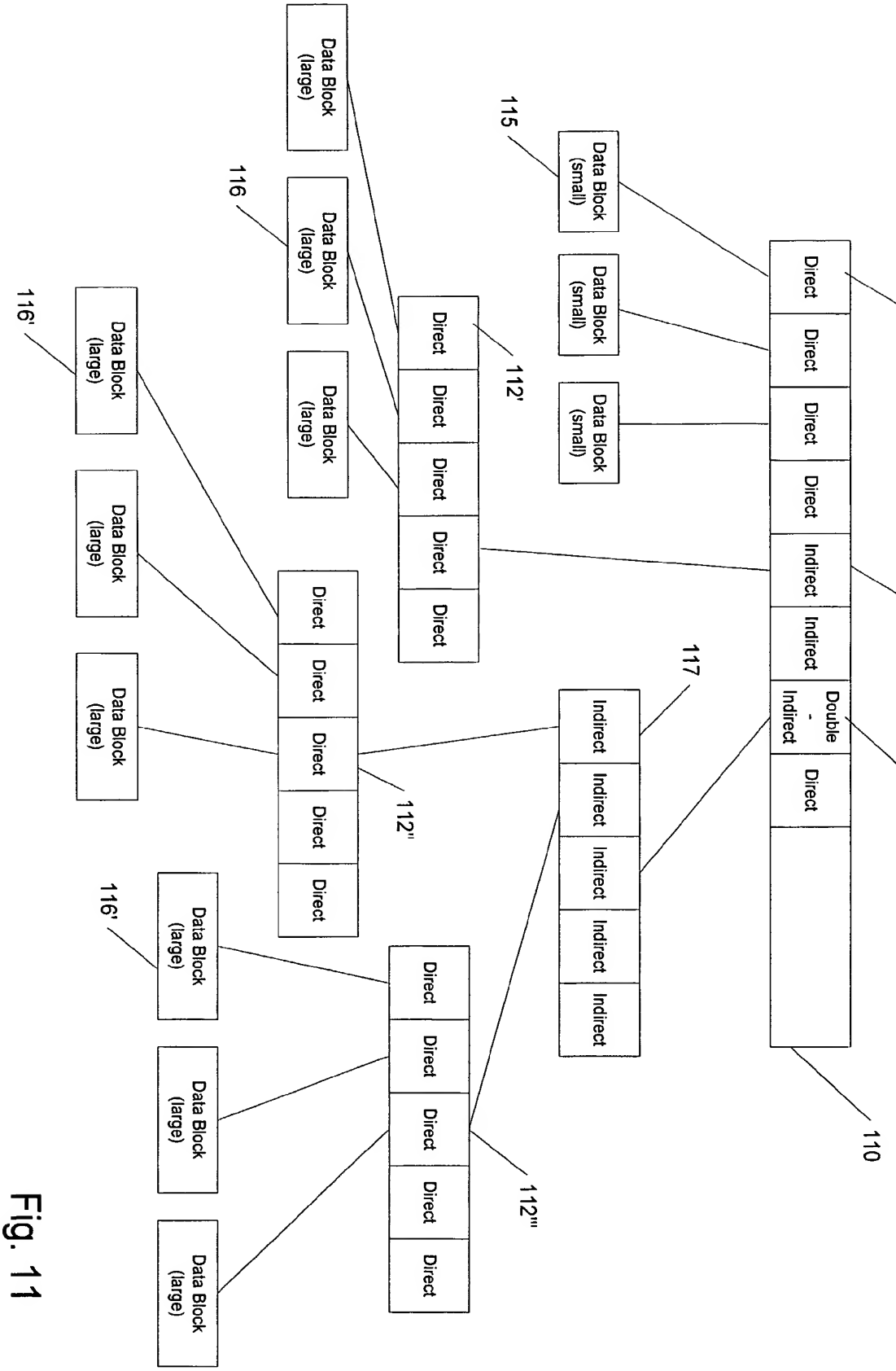

112
Direct
Direct
Direct
Direct
Indirect
Indirect
Double - Indirect
Direct
113
114
110
115
Data Block (small)
Data Block (small)
Data Block (small)
112'
Direct
Direct
Direct
Direct
Direct
116
Data Block (large)
Data Block (large)
Data Block (large)
117
Indirect
Indirect
Indirect
Indirect
Indirect
112''
Direct
Direct
Direct
Direct
Direct
116'
Data Block (large)
Data Block (large)
Data Block (large)
116'
Data Block (large)
Data Block (large)
Data Block (large)
112'''
Direct
Direct
Direct
Direct
Direct

Fig. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/12430

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : H04N 7/10; G06F 12/00
US CL : 709/ 217, 219
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/ 217, 219; 711/147, 150, 168; 707/10, 104, 231

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US 6,115,740 A (MIZUTANI) 05 September 2000 (05.09.2000), column 3, lines 46-67, col. 4, lines 1-29; column 8, lines 21-26; column12, lines 1-25.	1, 17 ---------- 2-16, 18- 85
Y	US 5,893,150 A (HAGERSTEN et al.) 06 April 1999 (06.04.1999), column 1, lines 50 -67; column 2, lines 1-21; column 6, lines 20 -46.	2-16, 18-85
Y	US 6,167,496 A (FECHNER) 26 December 2000 (26.12.2000), column 2, lines 36-55.	2-16, 18-85

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 July 2002 (16.07.2002)	09 AUG 2002
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Meng-Ai An for James R. Matthews Telephone No. (703) 305-3900

Form PCT/ISA/210 (second sheet) (July 1998)